LEARNING MADE EASY

2nd Edition

Logic

for dummies®

A Wiley Brand

- Understand logical operators and quantifiers
- Evaluate statements with truth tables
- Construct formal proofs of logical arguments

Mark Zegarelli

Author of *Fractions For Dummies* and *Calculus II For Dummies*

Logic

for dummies®

A Wiley Brand

Logic

2nd Edition

by Mark Zegarelli

for dummies®
A Wiley Brand

Logic For Dummies®, 2nd Edition

Published by: **John Wiley & Sons, Inc.,** 111 River Street, Hoboken, NJ 07030-5774, www.wiley.com

Copyright © 2026 by John Wiley & Sons, Inc. All rights reserved, including rights for text and data mining and training of artificial technologies or similar technologies.

Media and software compilation copyright © 2026 by John Wiley & Sons, Inc. All rights reserved, including rights for text and data mining and training of artificial technologies or similar technologies.

Published simultaneously in Canada

No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning or otherwise, except as permitted under Sections 107 or 108 of the 1976 United States Copyright Act, without the prior written permission of the Publisher or authorization through payment of the appropriate per-copy fee to the Copyright Clearance Center, Inc., 222 Rosewood Drive, Danvers, MA 01923, (978) 750-8400, fax (978) 750-4470, or on the web at www.copyright.com. Requests to the Publisher for permission should be addressed to the Permissions Department, John Wiley & Sons, Inc., 111 River Street, Hoboken, NJ 07030, (201) 748-6011, fax (201) 748-6008, or online at http://www.wiley.com/go/permissions.

The manufacturer's authorized representative according to the EU General Product Safety Regulation is Wiley-VCH GmbH, Boschstr. 12, 69469 Weinheim, Germany, e-mail: Product_Safety@wiley.com.

Trademarks: Wiley, For Dummies, the Dummies Man logo, Dummies.com, Making Everything Easier, and related trade dress are trademarks or registered trademarks of John Wiley & Sons, Inc. and may not be used without written permission. All other trademarks are the property of their respective owners. John Wiley & Sons, Inc. is not associated with any product or vendor mentioned in this book.

LIMIT OF LIABILITY/DISCLAIMER OF WARRANTY: THE PUBLISHER AND THE AUTHOR MAKE NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE ACCURACY OR COMPLETENESS OF THE CONTENTS OF THIS WORK AND SPECIFICALLY DISCLAIM ALL WARRANTIES, INCLUDING WITHOUT LIMITATION WARRANTIES OF FITNESS FOR A PARTICULAR PURPOSE. NO WARRANTY MAY BE CREATED OR EXTENDED BY SALES OR PROMOTIONAL MATERIALS. THE ADVICE AND STRATEGIES CONTAINED HEREIN MAY NOT BE SUITABLE FOR EVERY SITUATION. THIS WORK IS SOLD WITH THE UNDERSTANDING THAT THE PUBLISHER IS NOT ENGAGED IN RENDERING LEGAL, ACCOUNTING, OR OTHER PROFESSIONAL SERVICES. IF PROFESSIONAL ASSISTANCE IS REQUIRED, THE SERVICES OF A COMPETENT PROFESSIONAL PERSON SHOULD BE SOUGHT. NEITHER THE PUBLISHER NOR THE AUTHOR SHALL BE LIABLE FOR DAMAGES ARISING HEREFROM. THE FACT THAT AN ORGANIZATION OR WEBSITE IS REFERRED TO IN THIS WORK AS A CITATION AND/OR A POTENTIAL SOURCE OF FURTHER INFORMATION DOES NOT MEAN THAT THE AUTHOR OR THE PUBLISHER ENDORSES THE INFORMATION THE ORGANIZATION OR WEBSITE MAY PROVIDE OR RECOMMENDATIONS IT MAY MAKE. FURTHER, READERS SHOULD BE AWARE THAT INTERNET WEBSITES LISTED IN THIS WORK MAY HAVE CHANGED OR DISAPPEARED BETWEEN WHEN THIS WORK WAS WRITTEN AND WHEN IT IS READ.

For general information on our other products and services, please contact our Customer Care Department within the U.S. at 877-762-2974, outside the U.S. at 317-572-3993, or fax 317-572-4002. For technical support, please visit https://hub.wiley.com/community/support/dummies.

Wiley publishes in a variety of print and electronic formats and by print-on-demand. Some material included with standard print versions of this book may not be included in e-books or in print-on-demand. If this book refers to media that is not included in the version you purchased, you may download this material at http://booksupport.wiley.com. For more information about Wiley products, visit www.wiley.com.

Library of Congress Control Number: 2025945505

ISBN 978-1-394-36234-9 (pbk); ISBN 978-1-394-36236-3 (ePDF); ISBN 978-1-394-36235-6 (ePub)

Contents at a Glance

Introduction 1

Part 1: Getting Started with Logic 5

CHAPTER 1: Taking a Logical Perspective 7

CHAPTER 2: Following Logical Developments from Aristotle to AI 17

CHAPTER 3: Just for the Sake of Argument 35

Part 2: Flourishing with Formal Sentential Logic 61

CHAPTER 4: Engaging with Formal Affairs 63

CHAPTER 5: Embracing the Value of Evaluation 87

CHAPTER 6: Turning the Tables: Evaluating Statements with Truth Tables 101

CHAPTER 7: Taking the Easy Way Out: Creating Quick Tables 121

CHAPTER 8: Knowing How Truth Grows on Trees 137

Part 3: Exploring Proofs, Syntax, and Semantics in SL 159

CHAPTER 9: Providing Proof for What You Propose 161

CHAPTER 10: Applying Equivalence Rules 177

CHAPTER 11: Making Big Assumptions with Conditional and Indirect Proof 195

CHAPTER 12: Polishing Off Your Proofs 207

CHAPTER 13: Putting Operators in Their Place 229

CHAPTER 14: Syntactical Maneuvers and Semantic Considerations 237

Part 4: Questing After Quantification Logic 247

CHAPTER 15: Expressing Quantity with Quality: Quantification Logic 249

CHAPTER 16: Embracing QL Translations 263

CHAPTER 17: Proving Arguments with QL 275

CHAPTER 18: Expanding Logical Relations 301

CHAPTER 19: Planting a Quantity of Truth Trees 315

Part 5: Meditating on Modern Developments in Logic 327

CHAPTER 20: Computer Logic 329

CHAPTER 21: Sporting Propositions: Non-classical Logic 341

CHAPTER 22: Paradox and Axiomatic Systems 357

Part 6: The Part of Tens 369
CHAPTER 23: Ten (or So) Quotes about Logic 371
CHAPTER 24: Ten Big Names in Logic 373
CHAPTER 25: Ten Tips for Passing a Logic Exam 377

Index 383

Table of Contents

INTRODUCTION 1
About This Book 1
Foolish Assumptions 3
Icons Used in This Book 3
Beyond the Book 4
Where to Go from Here 4

PART 1: GETTING STARTED WITH LOGIC 5

CHAPTER 1: Taking a Logical Perspective 7
Getting an Overview of Practical Logic 8
Bridging the gap from here to there 8
Understanding cause and effect 9
Knowing everything and more 10
Explaining existence itself 11
Identifying a few logical words 11
Building and Evaluating Logical Arguments 12
Generating premises 12
Bridging the gap with intermediate steps 13
Forming a conclusion 13
Deciding whether the argument is valid 13
Understanding enthymemes (assumptions) 13
Making Logical Conclusions Simple with the Laws of Thought 14
The law of identity 14
The law of the excluded middle 15
The law of non-contradiction 15
Combining Logic and Math 16
Math is good for understanding logic 16
Logic is good for understanding math 16

CHAPTER 2: Following Logical Developments from Aristotle to AI 17
Classical Logic — from Aristotle to the Enlightenment 18
Aristotle invents syllogistic logic 18
Euclid's axioms and theorems 22
Chrysippus and the Stoics 22
Medieval logic and the extension of Aristotelian logic 23

Modern Logic — the 17th, 18th, and 19th Centuries 26
Leibniz and the Renaissance 26
Working up to formal logic 27
Logic in the 20th Century and Beyond 31
Non-classical logic 31
Gödel's proof 32
The age of computers 32
Artificial intelligence 33
Searching for the final frontier 33

CHAPTER 3: **Just for the Sake of Argument** 35
Defining Logic as it Relates to Arguments 36
Examining argument structure 36
Looking for validation 38
Studying Examples of Arguments 39
Ice cream Sunday 40
Key question 40
New York state of mind 41
Thinking Again: Logical Fallacies 41
Fallacies of relevance 42
Fallacies of weak induction 43
Fallacies of presumption 45
Fallacies of unwarranted assumptions 47
Fallacies of ambiguity 48
Understanding What Logic Is and Isn't 50
Thinking versus logic 51
Reality — what a concept! 52
The sound of soundness 53
Deduction and induction 54
Rhetorical questions 55
Whose Logic Is It, Anyway? 57
Pick a number (math) 57
Fly me to the moon (science) 57
Boot up or power down (computer science) 58
Tell it to the judge (law) 58
Find the meaning of life (philosophy) 59

PART 2: FLOURISHING WITH FORMAL SENTENTIAL LOGIC 61

CHAPTER 4: **Engaging with Formal Affairs** 63
Observing the Formalities of Sentential Logic 64
Turning language into statement constants 64
Having a truth value for every statement 65
Deploying the Five SL Operators 65
Feeling negative 66
Displaying a show of ands 68

Digging for or 69
Getting iffy 71
Getting even iffier 74
Seeing How SL Is Like Simple Arithmetic 75
The ins and outs of values 76
There's no substitute for substitution 77
Parenthetical guidance suggested 77
(Not) Getting Lost in Translation 78
The easy way — translating from SL to English 79
The not-so-easy way — translating from English to SL 81

CHAPTER 5: **Embracing the Value of Evaluation** 87
Evaluating Is the Bottom Line 88
Noticing how logic is similar to math 88
Getting started with SL evaluation 89
Stacking up another method 90
Making a Statement 92
Identifying sub-statements 92
Scoping out a statement. 93
The main attraction: Finding the main operator of a statement 94
Recognizing the Eight Forms of SL Statements 96
Revisiting Evaluation 98

CHAPTER 6: **Turning the Tables: Evaluating Statements with Truth Tables** 101
Putting It All on the Table: The Joy of Brute Force 102
Constructing Baby's First Truth Table. 103
Setting up a truth table. 103
Filling in a truth table 105
Reading a truth table 108
Putting Truth Tables to Work. 108
Taking on tautologies and contradictions 109
Judging semantic equivalence 109
Staying consistent across statements 111
Arguing with validity 113
Building New Tautologies and Contradictions 116
Turning a tautology into a contradiction, and vice versa. 117
Changing a pair of semantically equivalent statements into a tautology 117
Converting a set of inconsistent statements into a contradiction 118
Recasting a valid argument as a contradiction 118

CHAPTER 7: **Taking the Easy Way Out: Creating Quick Tables** 121
Dumping the Truth Table for a New Friend: the Quick Table 122
Considering a new approach to evaluating SL statements 122
Recognizing that a quick table may suffice 123
Outlining the Quick Table Process 123
Making a strategic assumption 124
Filling in a quick table 125
Reading a quick table 125
Disproving the assumption 126
Planning Your Strategy 127
Tautology 127
Contradiction 127
Contingent statement 128
Semantic equivalence and inequivalence 128
Consistency and inconsistency 128
Validity and invalidity 129
Working Smarter (Not Harder) with Quick Tables 130
Recognizing the six easiest types of statements to work with . . . 130
Working with the four not-so-easy statement types 132
Coping with the six difficult statement types 134

CHAPTER 8: **Knowing How Truth Grows on Trees** 137
Understanding How Truth Trees Work 137
Decomposing SL statements 138
Solving problems with truth trees. 140
Showing Consistency or Inconsistency. 141
Constructing the truth tree 141
Determining consistency or inconsistency from the truth tree 143
Testing for Validity or Invalidity 144
Choosing the order for decomposing 144
Following the order 145
Checking the validity or invalidity 146
Separating Tautologies, Contradictions, and Contingent Statements 147
Tautologies 147
Contradictions 151
Contingent statements 153
Checking for Semantic Equivalence or Inequivalence 153
Creating the first truth tree 154
Creating the second tree (if you need to) 156

PART 3: EXPLORING PROOFS, SYNTAX, AND SEMANTICS IN SL 159

CHAPTER 9: **Providing Proof for What You Propose** 161
Bridging the Premises-Conclusion Divide 162
Engaging the Eight Implication Rules in SL 163
The If-Rules: Modus Ponens and Modus Tollens 164
Modus ponens 164
Modus tollens 166
The And-Rules: Conjunction and Simplification 167
Conjunction 167
Simplification 169
The Or-Rules: Addition and Disjunctive Syllogism 170
Addition 170
Disjunctive syllogism 171
The Double-If-Rules: Hypothetical Syllogism and Constructive Dilemma 173
Hypothetical syllogism 173
Constructive dilemma 174

CHAPTER 10: **Applying Equivalence Rules** 177
Distinguishing Implications from Equivalences 178
Thinking of equivalences as ambidextrous 178
Applying equivalences to part of the whole 178
Discovering the Ten Valid Equivalences 179
Double negation 179
Contraposition 180
Implication 181
Exportation 183
Commutation 184
Association 185
Distribution 186
De Morgan's theorem 188
Tautology 190
Equivalence 191

CHAPTER 11: **Making Big Assumptions with Conditional and Indirect Proof** 195
Conditioning Your Premises with Conditional Proof 196
Understanding conditional proof 197
Tweaking the conclusion 199
Stacking assumptions 201
Thinking Indirectly: Verifying Arguments with Indirect Proof 202
Introducing indirect proof 203
Proving short conclusions 204
Combining Conditional and Indirect Proof 205

CHAPTER 12: **Polishing Off Your Proofs** 207
Easy Proofs: Taking a Gut Approach 208
Looking at the problem 208
Jotting down the easy stuff 209
Knowing when to move on 211
Moderate Proofs: Recognizing When to Use Conditional Proof 211
The three friendly forms: $x \rightarrow y$, $x \vee y$, and $\sim (x \,\&\, y)$ 212
The two slightly-less-friendly forms: $x \leftrightarrow y$ and $\sim (x \leftrightarrow y)$ 214
The three unfriendly forms: $x \,\&\, y$, $\sim (x \rightarrow y)$, and $\sim (x \vee y)$ 216
Difficult Proofs: Figuring Out What to Do When the Going Gets Tough 216
Choosing carefully between direct and indirect proof 216
Working backward from the conclusion 218
Digging deeper into SL statements 220
Breaking down long premises 224
Making a shrewd assumption 226

CHAPTER 13: **Putting Operators in Their Place** 229
Making Do with the Five SL Operators 230
Downsizing the Operators — a Semi-true Story 231
The tyranny of power 232
The blow of insurrection 233
The horns of dilemma 233
The (Sheffer) stroke of genius 234
The moral of the story 235

CHAPTER 14: **Syntactical Maneuvers and Semantic Considerations** 237
Working WFF Us or Against Us 238
Understanding WFFs (with a few strings attached) 239
Relaxing the rules 240
Separating WFFs from non-WFFs 241
Comparing SL to Boolean Algebra 242
Reading the signs 242
Doing the math 244
Running rings around things 245
Exploring syntax and semantics in Boolean algebra 245

PART 4: QUESTING AFTER QUANTIFICATION LOGIC 247

CHAPTER 15: **Expressing Quantity with Quality: Quantification Logic** 249
Taking a Quick Look at Quantification Logic 250
Identifying individual constants and property constants 251
Incorporating the SL operators 253
Understanding individual variables 254

Expressing Quantity with Two New Operators 255
Using the universal quantifier 255
Conveying existence with the existential quantifier 256
Creating context with the domain of discourse 257
Picking Out Statements and Statement Forms 259
Determining the scope of a quantifier 260
Discovering bound variables and free variables 260
Knowing the difference between statements and statement forms 261

CHAPTER 16: **Embracing QL Translations** 263
Translating the Four Basic Forms of Categorical Statements 264
Working with "all" and "some" 264
Moving on to "not all" and "no" 267
Discovering Alternative Translations of the Basic Forms 268
Translating "all" with ∃ 269
Interpreting "some" with ∀ 270
Converting "not all" with ∃ 270
Transforming "no" with ∀ 271
Identifying Statements in Disguise 271
Distinguishing "all" statements 272
Spotting "some" statements 272
Noticing "not all" statements 273
Recognizing "no" statements 273

CHAPTER 17: **Proving Arguments with QL** 275
Applying SL Rules in QL 276
Comparing similar SL and QL statements 276
Transferring the eight implication rules from SL into QL 277
Employing the ten SL equivalence rules in QL 279
Transforming Statements with Quantifier Negation 280
Introducing QN 281
Using QN in proofs 282
Exploring the Four Quantifier Rules 283
Easy rule #1: Universal instantiation 284
Easy rule #2: Existential generalization 287
Not-so-easy rule #1: Existential instantiation 290
Not-so-easy rule #2: Universal generalization 295

CHAPTER 18: **Expanding Logical Relations** 301
Relating to QL Relations 302
Defining and using relations 302
Connecting relational expressions 303
Making use of quantifiers with relations 304
Working with multiple quantifiers 305
Writing proofs with relations 307

Identifying Constants with Identities 310
Understanding identities 310
Writing proofs with identities 312

CHAPTER 19: **Planting a Quantity of Truth Trees** 315
Applying Your Truth Tree Knowledge to QL 315
Using the decomposition rules from SL 316
Adding UI, EI, and QN 317
Applying UI more than once 319
Reaching for the Sky: Non-Terminating Trees 323

PART 5: MEDITATING ON MODERN DEVELOPMENTS IN LOGIC 327

CHAPTER 20: **Computer Logic** 329
Remembering the Early Versions of Computers 330
Babbage designs the first computers 330
Turing and his universal Turing machine 331
Decoding the Modern Age of Computers 333
Getting the basics of hardware and logic gates 333
Comprehending software and computer languages 336
Ushering in Artificial Intelligence 337
Natural language: Speaking to machines on human terms 338
Problem-solving: Handling greater breadth and depth 338
Creative output: Mimicking human expression 339

CHAPTER 21: **Sporting Propositions: Non-classical Logic** 341
Opening Up to Possibility 342
Three-valued logic 342
Multi-valued logic 344
Fuzzy logic 345
Getting into a New Modality 347
Taking Logic to a Higher Order 349
Moving Beyond Consistency 351
Making Logical Sense of Quantum Logic 352
Breaking the law of the excluded middle 352
Looking at quantum logic differences 353
Disturbing the distribution rule 354

CHAPTER 22: **Paradox and Axiomatic Systems** 357
Grounding Logic in Set Theory 358
Setting things up 358
Trouble in paradox: Recognizing the problem with set theory 359
Developing a solution in Principia Mathematica 360
Discovering the Axiomatic System for SL 361

Proving Consistency and Completeness 363
Consistency and completeness of SL and QL 363
Formalizing logic and mathematics with Hilbert's Program 364
Examining Gödel's Incompleteness Theorem 365
The importance of Gödel's theorem 366
How he did it 366
Pondering the Meaning of It All 367

PART 6: THE PART OF TENS 369

CHAPTER 23: Ten (or So) Quotes about Logic 371
Ambrose Bierce 371
Samuel Butler 371
G. K. Chesterton 371
Albert Einstein 371
Joseph Wood Krutch 372
John Locke 372
Kate Mulgrew 372
Leonard Nimoy 372
Antoine de Saint-Exupéry 372
Gloria Steinem 372
Ludwig Wittgenstein 372

CHAPTER 24: Ten Big Names in Logic 373
Aristotle (384–322 BCE) 373
Peter Abelard (1079–1142) 374
Gottfried Leibniz (1646–1716) 374
George Boole (1815–1864) 374
Georg Cantor (1845–1918) 375
Gottlob Frege (1848–1925) 375
David Hilbert (1862–1943) 375
Bertrand Russell (1872–1970) 376
Kurt Gödel (1906–1978) 376
Alan Turing (1912–1954) 376

CHAPTER 25: Ten Tips for Passing a Logic Exam 377
Breathe 377
Start by Glancing over the Whole Exam 378
Warm Up with an Easy Problem First 378
Fill In Truth Tables Column by Column 378
If You Get Stuck on a Proof, Jot Down Everything 378
If You REALLY Get Stuck on a Proof, Move On 379

If Time Is Short, Finish the Tedious Stuff 379
Check Your Work 380
Admit Your Mistakes 380
Stay Focused Until the Bitter End 381

INDEX 383

Introduction

You use logic every day, and I bet you don't even realize it. For instance, consider these examples of common situations in which you may use logic:

- Planning an evening out with a friend
- Asking your boss for a day off or for a raise
- Picking out a shirt to buy among several that you like
- Explaining to your kids why homework comes before TV

All these scenarios require you to use logic to clarify your thinking and get other people to see things from your perspective.

Even if you don't always act on it, logic is natural, at least to humans. And logic is one of the big reasons humans have lasted so long on a planet filled with lots of other creatures that are bigger, faster, more numerous, and more ferocious. Because logic is already a part of your life, after you notice it, you'll see it working (or *not* working) everywhere you look.

In this book, I show you how logic arises naturally in daily life. When you recognize this, you can figure out how to refine certain types of thinking down to their logical essence. Logic gives you the tools for working with what you already know (the premises) to get you to the next step (the conclusion). It's also great for helping you spot flaws in arguments, such as unsoundness, hidden assumptions, or just plain unclear thinking.

About This Book

Logic has been around a long time — almost 2,400 years and counting! The ancient Greek philosopher Aristotle was the first to systematically study logic and attempt to formalize related ideas, and many later philosophers followed his lead. With so many people (past and present) thinking and writing about logic, you may find it difficult to know where to begin. But never fear, I wrote this book with beginners in mind.

If you're taking an introductory course in logic, you can supplement your knowledge with the information in this book. I explain just about everything you're studying in class simply, with lots of step-by-step examples. At the same time, if you're just interested in understanding what logic is all about, this book is a great place to start.

Logic is one of the few areas of study taught in two different college departments: math and philosophy. The reason it can fit into two seemingly different categories has a historical basis:

- **Its philosophical origin:** Logic began with Aristotle and has been developed by philosophers for centuries.
- **Its transition to computation:** About 150 years ago, mathematicians found that logic was an essential tool for grounding their work as foundational math became more and more abstract.

One of the most important results of this overlap is *formal logic*, which takes ideas from philosophical logic and applies them in a mathematical framework. Formal logic is usually taught in philosophy departments as a purely computational (that is, mathematical) exercise.

When writing this book, I tried to balance both of these aspects of logic. Generally speaking, the book begins where logic began — with philosophy — and ends where it transitioned — in mathematics.

To help you navigate throughout this book, I use the following conventions:

- *Italics* are used for emphasis and to highlight new words and terms defined in the text. They're also used for constants and variables in equations.
- **Boldface** text indicates keywords in bulleted lists and also true (**T**) and false (**F**) values in equations and tables. It's also used for the 18 rules of inference in sentential logic (SL) and the 5 rules of inference in quantification logic (QL).
- Sidebars are shaded gray boxes that contain text that's interesting but not critical to your understanding of the chapter or topic.
- Twelve-point boldface text (**T** and **F**) is used in running examples of truth tables and quick tables to indicate information that's just been added. It's used in completed truth tables and quick tables to indicate the truth value of the entire statement.
- Parentheses are used throughout statements, instead of a combination of parentheses, brackets, and braces. Here's an example: $\sim((P \vee Q) \rightarrow \sim R)$

Foolish Assumptions

Logic For Dummies is for anybody who wants to know about logic: what it is, where it came from, why it was invented, and even where it may be going. In this book, I give you an overview of logic in its many forms and provide you with a solid base of knowledge to build upon.

Here are a few things I assume about you because of your interest in this book:

- You want to find out more about logic, whether you're taking a course or just curious.
- You can distinguish between true and false statements about commonly known facts, such as "George Washington was the first U.S. president," and "The Statue of Liberty is in Tallahassee."
- You understand simple arithmetic.
- You can grasp simple algebra, such as solving for x in the equation $7 - x = 5$.

Icons Used in This Book

Throughout this book, you find four icons that highlight different types of information:

I use this icon to point out the key ideas you need to know. Make sure you understand the information in these paragraphs before reading on.

This icon highlights helpful hints that show you the easy way to get things done. Try them out, especially if you're enrolled in a logic course.

Don't skip these icons! They draw your attention to common errors you want to avoid and help you recognize where logic traps are hiding so you don't take a wrong step and get caught.

This icon alerts you to interesting, but unnecessary, trivia that you can read or skip over as you like.

Beyond the Book

This book comes with a free access-anywhere Cheat Sheet that includes a quick summary of logic basics, including logical operators, truth tables, implication rules, equivalence rules, and more. To get this Cheat Sheet, simply go to `www.dummies.com` and type *Logic For Dummies Cheat Sheet* in the Search box.

Where to Go from Here

Feel free to skip around in this book as you like. When I discuss a new topic that depends on more basic ideas, I refer to the chapter where I introduced those basics. If you only need info on a certain topic, check out the Index or the Table of Contents — they'll point you in the right direction.

If you have some background in logic and already have a handle on the Part 1 stuff, you can jump forward to where the action is. Each part builds on the previous parts, so if you can understand the material in Part 3, you probably don't need to concentrate on Parts 1 and 2 (unless, of course, you just want a little review).

If you're taking a logic course, you may want to read Parts 3 and 4 carefully. You may even try to reproduce the proofs in those chapters with the book closed. Better to find out what you don't know while you're studying than while you're sweating out an exam!

If you forge ahead to Parts 4 and 5, you're probably ready to tackle some fairly advanced ideas. If you're itching to get to some meaty logic, check out Chapter 22. This chapter on logical paradoxes has some really cool stuff to take your thinking to warp speed. Bon voyage!

1 Getting Started with Logic

IN THIS PART . . .

See how you constantly use logic to turn the facts at hand into a better understanding of the world.

Get a perspective on the history of logic, with a look at various theories that have developed over the centuries.

Understand the basic structure of a logical argument, focus on key concepts such as the premises and the conclusion, and discover how to identify and counter a variety of logical fallacies.

IN THIS CHAPTER

- Seeing the world from a logical point of view
- Using logic to build valid arguments
- Applying the laws of thought
- Understanding the connections between math and logic

Chapter 1
Taking a Logical Perspective

You and I live in an illogical world. If you doubt this fact, just watch a few YouTube videos. Or really listen to the person sitting at the next barstool. Or, better yet, spend the weekend with your in-laws.

With so many people thinking and acting illogically, why should you be any different? Wouldn't it be more sensible just to be as illogical as the rest of the human race?

Well, okay, being illogical *on purpose* is probably not the best idea. For one thing, how can trying to be illogical possibly be sensible? For another, if you've picked up this book in the first place, you probably aren't built to be illogical. Let's face it — some folks thrive on chaos (or claim to), and others don't.

In this chapter, I introduce you to the basics of logic and how it applies to life. I tell you about a few words and ideas that are key to logic. And I touch briefly on the connections between logic and math.

Getting an Overview of Practical Logic

Whether you know it or not, you already understand a lot about logic. In fact, you already have a built-in logic detector. Don't believe me? Take this quick test to see whether you're logical:

> **Q:** How many pancakes does it take to shingle a doghouse?
>
> **A:** 23, because bananas don't have bones.

If the answer seems illogical to you, that's a good sign that you're at least on your way to being logical. Why? Simply put, if you can spot something that's illogical, you must have a decent sense of what's actually logical.

In this section, I start with what you *already* understand about logic (though you may not be aware of that knowledge) and build toward a foundation that can help you in your study of logic.

Bridging the gap from here to there

Most children are naturally curious. They constantly want to know *why* everything is the way it is. And for every *because* answer they receive, they have one more *why* question. For example, consider these common kid questions:

> Why does the sun rise in the morning?
>
> Why do I have to go to school?
>
> Why does the car start when you turn the key?
>
> Why do people break the law when they know they could go to jail?

When you think about it, every great mystery seeks a solution via these types of questions: Even when the world doesn't make sense on its own, why does it feel like it should?

Kids sense from an early age that even though they don't understand something — like why the sun rises in the morning — the answer must be somewhere. And they think, "If I'm *here* and the answer is somewhere else, what do I have to do to get *there*?" (Often, their determination to get there leads them to bug their parents with more questions.)

REMEMBER

Human desire to get from here to there — from ignorance to understanding — is one of the main reasons logic came into existence as philosophy and science. Logic grew out of an essential human need to make sense of the world and, as much as possible, gain some control over it.

Understanding cause and effect

One way to understand the world is to notice the connection between cause and effect.

As you grow from a child to an adult, you begin to piece together how one event causes another. Typically, you can express the connections between cause and effect by using an *if-statement* (which is a basic conditional structure in logic). For example, consider these if-statements:

> *If* I let my favorite ball roll under the couch, *then* I can't reach it.
>
> *If* I do all my homework before Dad comes home, *then* he'll play catch with me before dinner.
>
> *If* I practice on my own this summer, *then* the coach will pick me for the team in the fall.
>
> *If* I keep showing up at job interviews prepared and with confidence, *then* I'll eventually get a job.

TIP

Understanding how if-statements work is an important aspect of logic.

Breaking down if-statements

Every if-statement is made up of two smaller statements called *sub-statements*: The *antecedent*, which follows the word *if*, and the *consequent*, which follows the word *then*. For example, consider this if-statement:

> *If* it is 5 p.m., *then* it's time to go home.

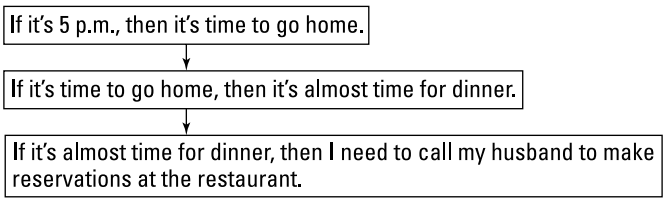
In this if-statement, the antecedent is the sub-statement *it is 5 p.m.* The consequent is the sub-statement *it's time to go home*.

REMEMBER

Every sub-statement can stand as a complete statement in its own right.

Stringing if-statements together

In many cases, the consequent of one if-statement becomes the antecedent of another. When this happens, you get a string of consequences, which the Greeks called a *sorites* (suh-*rye*-tease). For example:


If it's 5 p.m., then it's time to go home.
If it's time to go home, then it's almost time for dinner.
If it's almost time for dinner, then I need to call my husband to make reservations at the restaurant.

In this case, you can link these if-statements together to form a new if-statement:

> If it's 5 p.m., then I need to call my husband to make reservations at the restaurant.

Thickening the plot

As you gain more life experience, you may find that the connections between cause and effect become more and more sophisticated:

> *If* I let my favorite ball roll under the couch, *then* I can't reach it, *unless* I scream so loud that Grandma gets it for me, *though if* I do that more than once, *then* she gets annoyed and puts me back in my highchair.
>
> *If* I practice on my own this summer *but* not so hard that I blow my knees out, *then* the coach will pick me for the team in the fall *only if* there's a position open, *but if* I do *not* practice, *then* the coach will *not* pick me.

Knowing everything and more

As you begin to understand the world and what you find in it, you begin to make more general statements about it. For example:

> All horses are friendly.
>
> All 5-year-olds are silly.
>
> Every teacher at that school is out to get me.
>
> Every time I hear a phone ring in our house, it's my sister's phone.

REMEMBER

Words like *all* and *every* allow you to categorize things into *sets* (groups of objects) and *subsets* (groups within groups). For example, when you say, "All horses are friendly," you mean that the set of all horses is *contained within* the set of all friendly things — that is, horses are a *subset* of friendly things.

Explaining existence itself

You also discover connections within the world by figuring out what *exists* and *doesn't exist.* For example:

Some of my teachers are nice.

There is at least one student in school who likes me.

No one in the chess club can beat me.

There is no such thing as a Martian.

REMEMBER

Existence statements like these generally use wording that points to connections (or not):

- **An intersection of sets:** Words like *some*, *there is*, and *there exists* show an overlapping of sets called an *intersection.* For example, when you say, "Some of my teachers are nice," you mean that there's an intersection between the set of your teachers and the set of nice things.
- **No intersection between sets:** Words like *no*, *there is no*, and *none* show that there's no intersection between sets. For example, when you say, "No one in the chess club can beat me," you mean that there's no intersection between the set of all the chess club members and the set of all the chess players who can beat you.

Identifying a few logical words

As you can see, certain words show up a lot as you begin to make logical connections. Some of these common words are:

if . . . then	and	but	or
not	unless	though	every
all	only if	each	there is
there exists	some	there is no	none

Taking a closer look at words like these is an important step in the development of logic. When you do, you begin to see how these words enable you to divide and categorize the world in different ways (and therefore understand it better).

Building and Evaluating Logical Arguments

When people say "Let's be logical" about a given situation or problem, they often mean "Let's follow steps like these":

1. **Figure out what we know to be true.**
2. **Spend some time thinking about it.**
3. **Find the best course of action.**

REMEMBER

In logical terms, this three-step process involves building a *logical argument* — an explanation or rationale that contains a set of premises at the beginning and a conclusion at the end. In many cases, the premises and the conclusion will be linked by a series of intermediate steps. In the following sections, I discuss the steps in the order you're likely to encounter them.

Generating premises

The *premises* are the facts of the matter: the statements that you know or believe to be true. In many situations, writing down a set of premises is a great first step to problem-solving.

For example, suppose you're a school board member trying to decide whether to support the construction of a new school that would open for the school term that begins in September. Everyone is excited about the project, but you make some phone calls and piece together your facts to construct the following premises:

> The funds for the project won't be available until March.
>
> The construction company won't begin work until they receive payment.
>
> The entire project will take at least eight months to complete.

So far, you have only a set of three premises. But when you put these premises together, you get closer to the conclusion, which will complete your logical argument. In the next section, I show you how to combine the premises.

Bridging the gap with intermediate steps

Sometimes a logical argument is just a set of premises followed by a conclusion. In many cases, however, an argument also includes *intermediate steps* that show how the premises gradually lead to that conclusion.

Using the school construction example from the previous section, you may want to spell things out like this:

> According to the information we have, we won't be able to pay the construction company until March, so they won't be done until at least eight months later, which is November. But the school term begins in September. Therefore . . .

The word *therefore* indicates a conclusion and is the beginning of the final step, which I discuss in the next section.

Forming a conclusion

The *conclusion* is the outcome of your logical argument. If you've worked through the intermediate steps in a clear progression, the conclusion should be fairly obvious. For the school construction example I've been using, here it is:

> The building won't be complete before the school term begins in September.

If the conclusion doesn't make sense, something may be wrong with your argument. In some cases, an argument may not be valid. In others, you may have missing premises that you'll need to add.

Deciding whether the argument is valid

After you've built a logical argument, you need to be able to decide whether it's *valid*, which means that it's a good argument.

REMEMBER

To test a logical argument's validity, you assume that all the premises are true and then see whether the conclusion follows automatically from them. If the conclusion follows automatically, you know it's a valid argument. If not, the argument is *invalid*.

Understanding enthymemes (assumptions)

The school construction argument (from the example in the previous sections) may seem valid, but you also may have a few doubts. For example, if another

source of funding became available, the construction company could start earlier than March and perhaps finish by the September deadline. And so, the argument has a hidden premise, or assumption, called an *enthymeme* (*en*-thi-meem), as follows:

> There is no other source of funds for the project.

TIP

Logical arguments about real-world situations (in contrast to mathematical or scientific arguments) almost always have enthymemes — that is, hidden assumptions that may go unacknowledged. So, the clearer you understand the enthymemes hidden in an argument, the better chance you have of making sure your argument is valid.

Uncovering hidden premises in real-world arguments is more closely related to *rhetoric* — the study of how to make clear and convincing arguments — than it is to logic. I touch upon rhetoric, logical fallacies, and other details about the structure of logical arguments in Chapter 3.

Making Logical Conclusions Simple with the Laws of Thought

As a basis for understanding logic, philosopher Bertrand Russell set down three laws of thought. These laws are all grounded in ideas dating back to Aristotle, who founded classical logic more than 2,300 years ago. (See Chapter 2 for more on the history of logic.)

All three laws of thought are basic and easy to understand. The important thing to note is that these laws enable you to draw logical conclusions about statements even if you aren't familiar with the real-world circumstances they're discussing.

The law of identity

REMEMBER

The *law of identity* states that every individual thing is identical to itself.

For example:

> Jason Sudeikis is Jason Sudeikis.
>
> My cat, Ian, is my cat, Ian.
>
> The Washington Monument is the Washington Monument.

Without any information about the world, you can see from logic alone that all these statements are true. The law of identity tells you that any statement in the form of *X is X* must be true. In other words, any individual thing in the universe is the same as itself.

The law of the excluded middle

REMEMBER

The *law of the excluded middle* states that every statement is either true or false.

For example, consider these two statements:

> My first name is Mark.
>
> My first name is Algernon.

Again, without any information about the world, you know logically that each of these statements is either true or false. According to the law of the excluded middle, no third option is possible — in other words, statements can't be partially true or false. Rather, in logic, every statement is either completely true or completely false.

As it happens, I'm content that the first statement is true and relieved that the second is false.

The law of non-contradiction

REMEMBER

The *law of non-contradiction* states that given a statement and its opposite, one is true, and the other is false.

For example:

> My first name is Algernon.
>
> My first name is not Algernon.

Even if you don't know my name, you can be sure from logic alone that one of these statements is true and the other is false. In other words, because of the law of non-contradiction, my first name can't both be and not be Algernon.

Combining Logic and Math

Throughout this book, I often prove my points with examples that use math. (Don't worry — none of my examples is more complicated than what you learned in fifth grade or before.) Math and logic go great together for two reasons, which I explain in the following sections.

Math is good for understanding logic

Throughout this book, as I explain logic to you, I sometimes need examples that are clearly true or false to prove my points. As it turns out, math examples are great for this purpose because, in math, a statement is always either true or false, with no gray area between.

On the other hand, sometimes random facts about the world may be more subjective, or up for debate. For example, consider these two statements:

> Rihanna is an amazing singer.
>
> *Romeo and Juliet* is a lousy play.

Many people may well agree in this case that the first statement is true and the second is false, but both statements are up for debate. Now look at these two statements:

> The number 7 is less than the number 8.
>
> Five is an even number.

Clearly, there's no disputing that the first statement is true and the second is false.

Logic is good for understanding math

As I discuss in the section "Making Logical Conclusions Simple with the Laws of Thought" earlier in this chapter, the laws of thought on which logic is based — such as the law of the excluded middle — depend on black-and-white thinking. And, well, not many subjects are more black-and-white than math. Even though you may find studying history, literature, politics, or the arts to be more fun, they contain many more shades of gray.

Math is built on logic like a house is built on a foundation. If you're interested in the connection between math and logic, check out Chapter 22, which focuses on how math starts with obvious facts called *axioms* and then uses logic to form interesting and complex conclusions called *theorems.*

IN THIS CHAPTER

- Understanding the roots of logic
- Examining classical and modern logic
- Looking at 20th and 21st century developments in logic

Chapter 2

Following Logical Developments from Aristotle to AI

When you think about how *illogical* humans can be, you may be surprised to discover how much they've developed the concepts of logic over the years. Here's just a partial list of some varieties of logic that are floating around in the big world of premises and conclusions:

Boolean logic	Modern logic	Quantification logic
Classical logic	Multi-valued logic	Quantum logic
Formal logic	Non-classical logic	Sentential logic
Fuzzy logic	Predicate logic	Syllogistic logic
Informal logic	Propositional logic	Symbolic logic

As your eyes scan all these varieties of logic, you may feel a sudden urge to embrace your humanity fully and leave logic to the Vulcans. The good news, as you can soon discover, is that many types of logic are quite similar. After you're familiar with a few of them, the rest become much easier to understand.

So, where did all these types of logic come from? Well, that's a long story — in fact, it's a story that spans more than 2,000 years. I know 2,000 years seems like quite a lot to cram into one chapter, but I guide you through only the most important details. So, get ready for your short history lesson.

Classical Logic — from Aristotle to the Enlightenment

The ancient Greeks had a hand in discovering just about everything, and logic is no exception. For example, Thales and Pythagoras applied logical argument to mathematics. Socrates and Plato applied similar types of reasoning to philosophical questions. But the true founder of classical logic was Aristotle.

WARNING

When I talk about *classical logic* in this section, I'm referring to the historical period in which logic was developed, in contrast with *modern logic*, which I discuss later in the chapter. Classical logic, however, can also mean the most standard type of logic (which most of this book is about) in contrast with *non-classical logic* (which I discuss in Chapter 21). I try to keep the distinction clear as I go along.

Aristotle invents syllogistic logic

Before Aristotle (384–322 BCE), ancient scholars applied logical argument intuitively where appropriate in math, science, and philosophy. For example, given that all numbers are either even or odd, if you could show that a certain number wasn't even, you knew, then, that it must be odd. The Greeks excelled at this divide-and-conquer approach. They regularly used logic as a tool to examine the world.

Aristotle, however, was the first to recognize that the tool itself could be examined and developed. In six writings on logic — later assembled as a single work called *Organon*, which means "tool" — he analyzed how a logical argument functions. Aristotle hoped that logic, under his new formulation, would serve as a tool of thought that would help philosophers understand the world better. The system he developed, based on step-by-step logical arguments called *syllogisms*, is also known as *Aristotelian logic*.

TECHNICAL STUFF

Aristotle considered the goal of philosophy to be scientific knowledge, and saw the structure of scientific knowledge as fundamentally logical. Using geometry as his model, he noted that science consisted of proofs, proofs of syllogisms (which I discuss later in this section), syllogisms of statements, and statements of terms.

So, in *Organon*, he worked from the bottom upward: The first book, the *Categories*, deals with terms; the second, *On Interpretation*, with statements; the third, *Prior Analytics*, with syllogisms; and the fourth, *Posterior Analytics*, with proofs.

REMEMBER

Prior Analytics, the third book in *Organon* series, delves directly into what Aristotle called *syllogisms*, which are argument structures that, by their very design, appear to be indisputably valid.

The idea behind the syllogism was simple — so simple, in fact, that it had been taken for granted by philosophers and mathematicians until Aristotle noticed it. In a syllogism, the premises and conclusions fit together in such a way that, once you accept the premises as true, you must accept that the conclusion is true as well — regardless of the content of the actual argument being made.

For example, here's Aristotle's most famous syllogism:

Premises:

All men are mortal.

Socrates is a man.

Conclusion:

Socrates is mortal.

The following argument is similar in form to the first. And it's the form of the argument, not the content, that makes it indisputable. Once you accept the premises as true, the conclusion follows as equally true.

Premises:

All clowns are scary.

Bobo is a clown.

Conclusion:

Bobo is scary.

Categorizing categorical statements

Aristotle focused much of his attention on understanding what he called *categorical statements*, which are simply statements that talk about whole categories of objects or people. Furniture, chairs, birds, trees, red things, Avengers movies, and cities that begin with the letter *T* are all examples of categories.

In keeping with the law of the excluded middle (which I discuss in Chapter 1), everything is either in a particular category or not in it. For example, a red chair is in the category of furniture, chairs, and red things, but not in the category of birds, trees, Avengers movies, or cities that begin with the letter *T*.

REMEMBER

Aristotle broke categorical statements down into the following two types:

- **Universal statements,** which usually start with a word like *all* or *every* — words that tell you something about an entire category. Here's an example of a universal statement:

  All dogs are loyal.

  This statement relates to two categories and tells you that everything in the category of *dogs* is also in the category of *loyal things*. You can consider this a universal statement because it tells you that loyalty is a universal quality of dogs.

- **Particular statements,** which usually start with *some* or *at least one* — words or phrases that tell you about the existence of at least one example within a category. Here's an example of a particular statement:

  Some bears are dangerous.

  This statement tells you that at least one item in the category of *bears* is also in the category of *dangerous things*. This statement is considered a particular statement because it tells you that at least one particular bear is dangerous.

Understanding the square of oppositions

REMEMBER

The *square of oppositions* — a tool Aristotle developed for studying categorical statements — organizes the four basic forms of categorical statements that appear frequently in syllogisms. These four forms are based on the positive and negative forms of universal and particular statements.

Aristotle organized these four types of statements into a simple chart similar to Table 2-1. His most famous example was based on the observation that all humans are mortal. However, the example in the table is inspired by my sleeping cat.

TABLE 2-1 The Square of Oppositions

	Positive Forms	Negative Forms
Universal Forms	**A:** *All cats are sleeping.*	**E:** *No cats are sleeping.*
	There doesn't exist a cat that isn't sleeping.	All cats are not sleeping.
	No cats are not sleeping.	There isn't a cat that is sleeping.
	Every cat is sleeping.	There doesn't exist a sleeping cat.
Particular Forms	**I:** Some cats are sleeping.	**O:** Not all cats are sleeping.
	Not all cats are not sleeping.	Some cats are not sleeping.
	At least one cat is sleeping.	There is at least one cat that isn't sleeping.
	There exists a sleeping cat.	Not every cat is sleeping.

As you can see from the table, each type of statement expresses a different relationship between the category of cats and the category of sleeping things. In English, you can express each type of statement in a variety of ways. For example, under the positive forms for universal statements in Table 2-1, you find instances of the English wording that use a double negative to create a positive statement. I list a few of the many ways to state the same idea in the table, but many more are possible in each case.

REMEMBER

Aristotle noticed relationships among all these types of statements. The most important of these relationships is the contradictory relationship between the statements that are diagonal from each other in the table. With *contradictory pairs*, one statement is true and the other false.

For example, look at the **A** and **O** statements in Table 2-1. Clearly, if every cat in the world is sleeping at the moment, then **A** is true and **O** is false; otherwise, the situation is reversed. Similarly, look at the **E** and **I** statements. If every cat in the world is awake, then **E** is true and **I** is false; otherwise, the situation is reversed.

TECHNICAL STUFF

If you're wondering, the letters for the positive forms **A** and **I** supposedly come from the Latin word *AffIrmo*, which means "I affirm." Similarly, the letters for the negative forms **E** and **O** are said to come from the Latin word *nEgO*, which means "I deny." The source of these designations is unclear, but you can rule out Aristotle, who spoke Greek, not Latin.

Euclid's axioms and theorems

Although Euclid (c. 325–265 BCE) wasn't strictly a logician, his contributions to logic were undeniable.

Euclid is best known for his work in geometry, which is still called *Euclidean geometry* in his honor. His greatest contribution to this field was his logical organization of geometric principles into *axioms* and *theorems*.

Euclid began with five *axioms* (also called *postulates*) — true statements that he believed were simple and self-evident. From these axioms, he used logic to prove *theorems* — true statements that were more complex and not immediately obvious. In this way, he succeeded in proving the vast body of geometry that logically followed from the five axioms alone. Mathematicians still use his logical organization of statements into axioms and theorems. (For more on this topic, see Chapter 22.)

Euclid also used a logical method called *indirect proof*. In this method, you assume the opposite of what you want to prove and then show that this assumption leads to a conclusion that's obviously incorrect.

For example, a detective in a murder mystery may reason in the following indirect way:

> If the butler committed the murder, then he must have been in the house between 7 p.m. and 8 p.m. (which is when the murder occurred).
>
> But witnesses saw him in the city 20 miles away during that hour, so he couldn't have also been in the house.
>
> Therefore, the butler didn't commit the murder.

Indirect proof is also called *proof by contradiction* and *reductio ad absurdum*, which is Latin for "reduced to an absurdity." (Flip to Chapter 11 for more about how to use indirect proof.)

Chrysippus and the Stoics

While Aristotle's successors developed his work on the syllogistic logic of categorical statements, another Greek school of philosophy, the Stoics, took a different approach. They focused on *conditional statements*, which are statements that take the form *of an if-statement.* For example:

> *If* clouds are gathering in the west, *then* it will rain.

Most notable among these logicians was Chrysippus (279–206 BCE). He examined arguments using statements that were in this if-statement. For example:

Premises:

If clouds are gathering in the west, then it will rain.

Clouds are gathering in the west.

Conclusion:

It will rain.

Certainly, you can find connections between the Aristotelian and the Stoic approaches. Both approaches focused on sets of premises containing statements that, when true, tended to fit together in a way that forced the conclusion to be true as well. But friction between the two schools of thought caused logic to develop in two separate paths for more than a century — though over time they merged into a unified discipline.

Medieval logic and the extension of Aristotelian logic

After Aristotle, medieval logic — which flourished between the 5th and 15th centuries — extended and refined Aristotelian logic. Thinkers such as Boethius, Avicenna, Thomas Aquinas, and William of Ockham played pivotal roles in shaping logical theories that influenced philosophy and religion. They adapted Aristotle's ideas about reasoning and developed new ways to of ordering arguments, which helped lay the foundation for later advances in science and philosophy. Their work bridged classical and modern thought, keeping logical thinking alive in medieval education and beyond.

Boethius: Affirming Aristotelian logic

The Roman philosopher Boethius (c. 477–524) helped preserve and spread Aristotle's ideas on logic to the medieval world. He translated and explained Aristotle's works, such as *Categories* and *On Interpretation*, making them easier to understand. He also advanced on theories about how statements fit together in logical arguments. His book *The Consolation of Philosophy*, while not strictly about logic, used logical reasoning to explore deep philosophical questions (see the sidebar "Boethius and *The Consolation of Philosophy*").

His contributions ensured that Aristotelian logic remained a cornerstone of education in the Middle Ages.

BOETHIUS AND *THE CONSOLATION OF PHILOSOPHY*

In *The Consolation of Philosophy*, Boethius applied logical reasoning to profound philosophical questions, such as

- **The nature of fortune and fate:** He questioned why good people suffer while bad people prosper. Through logical argumentation, he reasoned that true happiness does not come from wealth, power, or fame but from inner virtue.
- **The problem of evil:** He examined how evil can exist in a world governed by a just and all-powerful God. He argued that evil people are actually powerless because they lack true goodness, which is the highest form of being.
- **Divine foreknowledge and free will:** He addressed whether God's knowledge of the future means that human actions are predetermined. He concluded that God exists outside of time and sees all events at once, which allows for free will while maintaining divine omniscience (universal knowledge).

Avicenna: Expanding Aristotelian logic

A prominent Persian philosopher, Avicenna (c. 980–1037) expanded Aristotle's logic by introducing ideas about necessity, possibility, and impossibility, which anticipated later developments in modal reasoning (see Chapter 21). He also worked on the relationship between subjects and predicates in statements, improving Aristotle's system of logic. His major work, *The Book of Healing*, deeply influenced both Islamic and European medieval philosophy. Avicenna's ideas helped shift logic toward a more systematic and scientific approach, and they influenced many later thinkers.

Peter Abelard: Going modal

Peter Abelard (1079–1142) was a French philosopher and pioneering medieval logician who helped move logic beyond traditional patterns and explore how statements and ideas really work. He studied how language creates meaning and how varying words and phrases can affect whether a statement is true or false. Abelard's focus on *semantics* — how statements express truth and meaning (see Chapter 14) — made him a forerunner of modern ideas in logic and philosophy of language.

Abelard was also one of the first to explore *modal logic* — the logic of possibility and necessity (see Chapter 21), using words such as *can* and *must* — which laid the

groundwork for later advances in that area. In his writings, especially *Sic et Non*, he compared conflicting views from earlier thinkers to show how clear thinking could solve problems.

Thomas Aquinas: Connecting faith and reason

Thomas Aquinas (1225–1274), an Italian philosopher and priest, combined Aristotle's logic with Christian theology, forming a comprehensive scholastic method that became a major part of later medieval philosophy. His most important work, *Summa Theologica*, used logical arguments to explain religious beliefs, showing that faith and reason could work together. Aquinas also deepened the understanding of analogy and the interpretation of religious language, helping theologians use logical reasoning more effectively in discussing divine matters. His influence was long-lasting, shaping the way scholars used logic in religious and philosophical discussions.

William of Ockham: Relating language and thought

English philosopher and theologian William of Ockham (c. 1287–1347) is best known for *Ockham's* (or *Occam's*) *razor*, the idea that the simplest explanation is usually the best. He challenged unnecessary assumptions in logic and supported *nominalism*, the idea that general concepts (like *tree* or *redness*) are just names, not real things. In his book *Summa Logicae*, he redefined how logical statements connect and developed ideas about how language and thought relate. Ockham's work influenced later philosophy, including the development of scientific reasoning.

Jean Buridan: Consequences and clear thinking

Jean Buridan (c. 1300 – 1358) was a French philosopher and central figure in late medieval logic. He is best known for formalizing the *theory of consequences* (how one statement can follow logically from another) and advancing Abelard's work on modal logic (statements of possibility and necessity that use words such as *can* and *must*).

Building on earlier ideas from medieval thinkers, Buridan went beyond simple patterns like syllogisms to improve how people could understand what makes an argument valid. He explained how words can stand for different things depending on the situation that they describe; this explanation was an early version of ideas we now use in modern logic. (See Chapter 3 for a look at how the meaning of words can vary, resulting in a variety of logical fallacies.) In books such as *Summulae de Dialectica*, he stressed how logical language must keep arguments clear and make sure that the conclusions truly follow from the starting points.

Instructors used Buridan's books for hundreds of years to teach students how to reason well. His clear thinking helped connect the old style of logic with the new ways of reasoning that developed in the modern era.

Modern Logic — the 17th, 18th, and 19th Centuries

In Europe, as the Age of Faith gradually gave way to the Age of Reason in the 16th and 17th centuries, thinkers became optimistic about finding answers to questions about the nature of the universe.

Even though scientists (such as Isaac Newton) and philosophers (such as René Descartes) continued to believe in God, they looked beyond church teachings to provide answers about how God's universe operated. When they did, they found that many of the mysteries of the world — such as the fall of an apple or the motion of the moon in space — could be explained and predicted by using mathematics. With this surge in scientific thought, logic became a fundamental tool of reason.

Leibniz and the Renaissance

Gottfried Leibniz (1646–1716) was arguably the greatest logician of the Renaissance in Europe. Like Aristotle, Leibniz saw the potential for logic to become an indispensable tool for understanding the world. He was the first logician to take Aristotle's work a significant step further by turning logical statements into symbols that could then be manipulated like numbers and equations. The result was the first crude attempt at *symbolic logic*.

In this way, Leibniz hoped logic would transform philosophy, politics, and even religion into pure calculation, providing a reliable method to answer all of life's mysteries with objectivity. In a famous quote from *The Art of Discovery* (1685), he says:

> The only way to rectify our reasonings is to make them as tangible as those of the Mathematicians, so that we can find our error at a glance, and when there are disputes among persons, we can simply say: "Let us calculate, without further ado, to see who is right."

Unfortunately, his dream of transforming all areas of life into calculation wasn't pursued by the generation of thinkers that followed him. His ideas were so far ahead of their time that they weren't recognized as important. After his death,

Leibniz's writings on logic as symbolic calculation gathered dust for almost 200 years. By the time they were rediscovered, logicians had already caught up with these ideas and surpassed them. As a result, Leibniz wasn't nearly as influential as he might have been during this crucial phase in the development of logic.

Working up to formal logic

For the most part, logic was studied *informally* — that is, without the use of symbols in place of words — into the early 19th century. Beginning with Leibniz, mathematicians and philosophers up to this time had improvised a wide variety of notations for common logical concepts. But these systems generally lacked any method for full-scale computation and calculation.

REMEMBER

By the end of the 19th century, however, mathematicians had developed *formal logic* — also called *symbolic logic* — in which computable symbols stand for words and statements. Three key contributors to formal logic were George Boole, Georg Cantor, and Gottlob Frege.

Boolean algebra

Named for its inventor, English mathematician and philosopher George Boole (1815–1864), *Boolean algebra* is the first fully fleshed-out system that handles logic as calculation. For this reason, it's considered the precursor to formal logic.

REMEMBER

Boolean algebra is similar to standard arithmetic in that it uses both numerical values and the familiar operations for addition and multiplication. Unlike arithmetic, however, only two numbers are used: 0 and 1, which signify *false* and *true*, respectively.

For example:

> Let *A* = Thomas Jefferson wrote the Declaration of Independence.
>
> Let *B* = Charli D'Amelio wrote the U.S. Constitution.

Because the first statement is true and the second is false (thank goodness!), you can say:

> *A* = 1 and *B* = 0

In Boolean algebra, addition is interpreted as *or*, so the statement

> Thomas Jefferson wrote the Declaration of Independence *or* Charli D'Amelio wrote the U.S. Constitution.

is translated as

$$A + B = 1 + 0 = 1$$

Because the Boolean equation evaluates to 1, the statement is true.

Similarly, multiplication is interpreted as *and*, so the statement

> Thomas Jefferson wrote the Declaration of Independence *and* Charli D'Amelio wrote the U.S. Constitution.

is translated as

$$A \times B = 1 \times 0 = 0$$

In this case, the Boolean equation evaluates to 0, so the statement is false.

As you can see, the calculation of values is remarkably similar to arithmetic. But the meaning behind the numbers is pure logic.

Check out Chapter 14 for more information about Boolean algebra.

Cantor's set theory

REMEMBER

Set theory, pioneered by German mathematician Georg Cantor (1845–1918) in the 1870s, was another foreshadowing of formal logic, but with far wider influence and usefulness than Boolean algebra.

Loosely defined, a *set* is just a collection of things, which may or may not have something in common. Here are a few examples:

{1, 2, 3, 4, 5, 6}

{Batman, Wonder Woman, Spiderman}

{Africa, Dua Lipa, November, Snoopy}

This simple construction is tremendously effective for characterizing important core ideas of logic. For example, consider this statement:

> All U.S. states that contain the letter *z* begin with the letter *A*.

You can verify this statement by identifying two sets: all states that contain the letter *z* and all states that begin with the letter *A.* Here are the sets:

Set 1: {Arizona} **Set 2:** {Alabama, Alaska, Arizona, Arkansas}

As you can see, every member of the first set is also a member of the second set. Thus, the first set is a *subset* of the second set, so the original statement is true.

Despite its apparent simplicity — or, rather, because of it — set theory would soon become the foundation of formal logic and, ultimately, of mathematics itself (see the section "Logic in the 20th Century and Beyond" later in the chapter for more information).

Frege's formal logic

REMEMBER

German mathematician and logician Gottlob Frege (1848–1925) developed the first real system of formal logic. The system he devised is really one logical system embedded within another. The smaller system, *sentential logic* — also known as *propositional logic* — uses letters to stand for simple statements, which are then linked together using symbols for five key concepts: *not*, *and*, *or*, *if*, and *if and only if*. For example:

Let *E* = Evelyn is at the movies.

Let *P* = Peter is at home.

These definitions allow you to take these two statements:

Evelyn is at the movies *and* Peter is at home.

If Evelyn is at the movies, *then* Peter is *not* at home.

and turn them into symbols as follows:

$E \& P$

$E \rightarrow \sim P$

In the first statement, the symbol & means *and*. In the second, the symbol → means *if . . . then*, and the symbol ~ means *not*.

I discuss sentential logic in more detail in Parts 2 and 3.

The larger system, *quantification logic* — also known as *predicate logic* — includes all the rules from sentential logic, but expands upon them. Quantification logic uses different letters to stand for the subject and the predicate of a simple statement. For example:

Let e = Evelyn

Let p = Peter

Let Mx = x is at the movies

Let Hx = x is at home

These definitions allow you to represent these two statements:

Evelyn is at the movies *and* Peter is at home.

If Evelyn is at the movies, *then* Peter is *not* at home.

as

$Me \,\&\, Hp$

$Me \rightarrow \sim Hp$

Quantification logic includes two additional symbols for *all* (∀) and *some* (∃), which allows you to represent the statements

Everyone is at the movies.

Someone is at home.

as

$\forall x\,[Mx]$

$\exists x\,[Hx]$

Quantification logic has the power to represent the four basic categorical statements from Aristotle's square of oppositions (see the section "Categorizing categorical statements" earlier in the chapter). In fact, in its most developed form, quantification logic is as powerful as all previous formulations of logic.

Check out Part 4 for more on quantification logic.

Logic in the 20th Century and Beyond

By the end of the 19th century, following Euclid's example (see "Euclid's axioms and theorems" earlier in this chapter), mathematicians sought to reduce everything in mathematics to a set of theorems logically dependent on a small number of axioms.

Gottlob Frege, the inventor of the first real system of formal logic, saw the possibility that mathematics itself could be derived from logic and set theory. Beginning with only a few axioms about sets, he showed that numbers and, ultimately, all of mathematics followed logically from these axioms.

REMEMBER

Frege's theory seemed to work well until British mathematician and philosopher Bertrand Russell (1872–1970) found a *paradox*, or an inconsistency, which occurs when a set may contain itself as a member. Within Frege's system, Russell saw that it was possible to create a set containing every set that *doesn't* contain itself as a member. The problem is that if this set contains itself, then it doesn't contain itself, and vice versa. This inconsistency became known as *Russell's paradox*. (I discuss Russell's paradox further in Chapter 22.)

Frege was devastated by the error, but Russell saw merit in his work. From 1910 to 1913, Russell and fellow mathematician and philosopher Alfred North Whitehead produced the three-volume *Principia Mathematica*, a reworking of Frege's ideas grounding mathematics in axioms of set theory and logic.

Non-classical logic

The project of reducing math and logic to a short list of axioms opens up an interesting question: What happens if you start with a different set of axioms?

One possibility, for example, is to allow a statement to be something other than either true or false. In other words, you can allow a statement to violate the law of the excluded middle (see Chapter 1). The deliberate violation of this law would have been unthinkable to the Greeks, but with logic formulated simply as a set of axioms, the possibility became available.

In 1917, Polish logician Jan Lukasiewicz (1878–1956) pioneered the first *multi-valued logic*, in which a statement may be not only *true* or *false*, but also *possible*. This system would be useful for classifying a statement such as

> In the year 2162, the Yankees will win the World Series.

REMEMBER

The introduction of the category *possible* to the values *true* and *false* was the first radical departure from *classical logic* — all the logic that had existed before — into a new area called *non-classical logic*. (You can find out more about forms of non-classical logic — including fuzzy logic and quantum logic — in Chapter 21.)

Gödel's proof

Principia Mathematica, the three-volume series written by Bertrand Russell and Alfred North Whitehead, firmly established logic as an essential foundation of math. However, more surprises were in store for logic.

With mathematics defined in terms of a set of axioms, the question as to whether this new system was both consistent and complete arose. That is, was it possible to use logic to derive every true statement about math from these axioms, and no false statements?

In 1931, Kurt Gödel (1906–1978) showed that infinitely many mathematical statements are true but can't be proven given the axioms of the *Principia*. He also showed that any attempt to reduce math to a consistent system of axioms produces the same result: infinitely many of mathematical truths, called *undecidable statements*, which aren't provable within that system.

This result, called the *incompleteness theorem*, established Gödel as one of the greatest mathematicians of the 20th century.

REMEMBER

In a sense, Gödel's incompleteness theorem provided a response to Gottfried Leibniz's hope that logic would someday provide a method for calculating answers to all of life's mysteries. The response, unfortunately, was a definitive *no!* Logic — at least in its current formulation — is insufficient to prove every mathematical truth, let alone every truth of this complex world. (See Chapter 22 for more about Gödel's proof and its implications.)

The age of computers

Rather than focus on what logic can't do, however, mathematicians and scientists have found endless ways to use logic. Foremost among these uses is the computer, which some experts (especially computer scientists) deem the greatest invention of the 20th century.

Hardware, the physical design of computer circuitry, uses *logic gates*, which mimic the basic functions of sentential logic — taking input in the form of electric current from one or two sources and outputting current only under given conditions.

For example, a *NOT gate* outputs current only when no current is flowing in from its one input. An *AND gate* outputs only when current is flowing in from both of its inputs. And finally, an *OR gate* outputs only when current is flowing in from at least one of its two inputs.

Software, the programs that direct the actions of the hardware, is all written in *computer languages*, such as Python, JavaScript, Java, C, or C++. Although all computer languages have their differences, each contains a core of similarities, including a set of key words from sentential logic, such as *and*, *or*, *if . . . then*, and so on. (Check out Chapter 20 for more on how logic is used in computer hardware and software).

Artificial intelligence

Recent advances in artificial intelligence (AI) have merged the disciplines of logic, computing, and natural language, building on centuries of logical thought. AI systems, particularly in machine learning and natural language processing, rely on formal logic principles to process and generate human language. Formal logic helps AI recognize patterns and make sense of language.

Modern AI also integrates probabilistic logic, which allows machines to make decisions under uncertainty. Large language models like ChatGPT or Copilot use vast datasets and statistical reasoning to understand and respond to human queries. As AI continues to evolve, the fusion of logic, computing, and language makes AI processing more connected to human thinking and, thus, more applicable to daily use. (See Chapter 20 for more on this important topic.)

Searching for the final frontier

Will logic ever be sufficient to describe all the subtleties of the human mind and the complexities of the universe? My guess is probably not. (***Spoiler alert:*** See Chapter 22 for a look at how mathematician Kurt Gödel used logic to demonstrate the limits of logic itself.)

But logic is an immensely powerful tool whose uses have only begun to be tapped into. And who knows? Even as I write this book, logicians are working to develop more expressive logical systems to expand the capabilities of mathematics and the sciences. Their efforts may yet produce inventions that surpass all current expectations and dreams.

IN THIS CHAPTER

- Describing logical arguments
- Breaking down the parts of a logical argument
- Identifying some common logical fallacies
- Separating logic from the pretenders
- Examining the wide-ranging applications of logic

Chapter 3
Just for the Sake of Argument

Simply put, logic is the study of how to tell a good argument from a bad one.

In daily life, most people use the word *argument* to describe anything from a tense disagreement to a shouting match. But an argument doesn't have to be heated or angry. The idea behind *logical argument* is simple: I want to convince you of something, so I point out some facts that you already agree with. Then, I show you how what I'm trying to prove follows naturally from these facts.

In this chapter, I explain what logic is and walk you through the elements of a logical argument. I show you a variety of common *logical fallacies* — that is, types of arguments that may mislead you into accepting them. I also provide you with lots of examples, show you what logic *isn't*, and run through the various fields that use logic.

After you've read this chapter, you may still find yourself on the wrong end of an argument, but at least you won't easily be accused of being illogical!

Defining Logic as it Relates to Arguments

Here's what you need to know about logic and its components:

- Logic is the study of *argument validity*, which is whether a logical argument is *valid* (good) or *invalid* (bad).
- An *argument*, in logic, is a set of one or more *premises* followed by a *conclusion*, which are often connected by one or more *intermediate statements*.
- An argument's premises and conclusion are always *statements* — that is, sentences that give information and are either true or false.
- In a *valid argument*, if all the premises are true, the conclusion must also be true.

REMEMBER

When you put the preceding descriptive bullet points together, here's what you get:

> *Logic* is the study of how to decide the circumstances in which a set of true premises leads to a conclusion that is also true.

That's it! As you work through this book, keep this definition in mind. Write it down on an index card and tape it to the mirror in your bathroom. Every topic in logic relates to this central idea in some way.

Examining argument structure

A *logical argument* must have one or more premises followed by a conclusion that flows from the premises. Here's an example of a logical argument between a man and a woman:

> *Nick:* I love you.
>
> *Mabel:* Yes, I know.
>
> *Nick:* And you love me.
>
> *Mabel:* True.
>
> *Nick:* And people who love each other should get married.
>
> *Mabel:* Okay.
>
> *Nick:* So we should get married.

This may not be the most romantic marriage proposal you've ever heard, but you get the idea. If Mabel really does agree with all three of Nick's statements, his argument should convince her to marry him.

Identifying argument components

Take a closer look at the structure of Nick's argument and you can see that it contains three premises and one conclusion. Boiling them down to the basics, Nick is saying:

Premises:

I love you.

You love me.

People who love each other should get married.

Conclusion:

We should get married.

REMEMBER

The premises and the conclusion of an argument all have one thing in common: They are statements. A *statement* is simply a sentence that gives information that can be either true or false.

Distinguishing statements from non-statements

The following are all statements because they provide information that can be either true or false:

- The capital of Mississippi is Jackson.
- Two plus two equals five.
- Your red dress is prettier than your blue one.
- Humans are just like dogs.

By contrast, the following are *not* statements:

- **A big blue Cadillac.** Because this is a phrase and not a complete sentence, it doesn't give you enough information to determine its true or false value.
- **Do you come here often?** A question like this is asking for information, not providing information.

- **Clean your room right now.** A command may tell you what to do, but it doesn't have a true or false value.
- **Golly!** This exclamation may express a sentiment, but it has no true or false value.

TIP

In logic, a statement may be either true or false. This is called the *truth value* of the statement. Because logicians deal with truth values all the time, they save on the ever-rising cost of ink by referring to the truth value as just the *value* of the statement. In this book, I use both terms, but they mean the same thing.

Determining a statement's truth value

Sometimes you can easily verify the truth value of a statement. The value of the statement *The capital of Mississippi is Jackson* is true because Mississippi's capital is, in fact, Jackson. And two plus two is four, not five, so the value of the statement *Two plus two equals five* is false.

In other cases, the truth value of a statement is more difficult to verify. For example, how do you decide whether one dress really is prettier than another, or whether humans really are just like dogs?

For the moment, don't worry about *how* you figure out whether a statement is true or false, or even *if* you can figure it out. I touch on this later in the chapter in "The sound of soundness."

Looking for validation

REMEMBER

In a good argument — or a *valid argument*, as logicians say — when all the premises are true, the conclusion must also be true.

Valid arguments are at the very heart of logic. Remember, in a logical argument, I want to convince you of something, so I point out facts that you already agree with (*premises*), and then show you that what I'm trying to prove (the *conclusion*) follows from these facts. If the argument is valid, it's airtight, so the conclusion inevitably follows from the premises.

For example, suppose your professor tells you, "Everyone who studied did well on my midterm. And you studied, so you did well." Break this statement down into premises and a conclusion to see what you get.

Premises:

If a student studied, then they did well on the midterm.

You studied.

Conclusion:

You did well on the midterm.

The preceding set of premises and conclusion is an example of a valid argument. You can see that if both premises are true, then the conclusion must be true as well because — as a well-structured argument — the conclusion *must* logically follow from the premises.

When this key element is missing — that is, when the conclusion doesn't necessarily follow from the premises — an argument is invalid even if all of its statements are true. For example, here is an invalid argument:

Premises:

George Washington was the first U.S. president.

Albert Einstein proposed the theory of relativity.

Conclusion:

Mikey Madison won an Academy Award for *Anora.*

All these statements happen to be true. But that doesn't mean the argument is valid. In this case, the argument is invalid because the conclusion does not necessarily follow from the premises.

Studying Examples of Arguments

Because arguments are so important to logic, in this section, I give you a few more examples so you can get a sense of how arguments work. Some of these examples have arguments in which the first premise is an *if-statement: If* something happens, *then* something else will happen. In others, this structure of implication is implied.

Aristotle, the first person to study the form of arguments, offered this example:

Premises:

All men are mortal.

Socrates is a man.

Conclusion:

Socrates is mortal.

He called this form of argument a *syllogism*. (See Chapter 2 for a closer look at early forms of logic developed by Aristotle and other thinkers.)

After you get the hang of how logical arguments work, the variations are endless. In Parts 2 through 5 of this book, you discover even more precise and useful ways to create, understand, and prove logical arguments. For now, though, these examples give you a taste of what valid arguments are all about.

Ice cream Sunday

Suppose that on Sunday, your son, Andrew, reminds you: "You said that if we go to the park on Sunday, we can get ice cream. And now we're going to the park, so that means we can get ice cream." His logic is impeccable. To show why this is so, here is Andrew's argument separated into premises and a conclusion:

Premises:

If we go to the park, then we can get ice cream.

We're going to the park.

Conclusion:

We can get ice cream.

The first premise sets up the if-then slope, and the second premise is where you step onto it. As a result, you land inevitably in the conclusion.

Key question

Suppose one afternoon, you arrive home from school to find your mother looking for her keys. You tell her: "You always leave your keys either in your purse or in the basket next to the door. They're not in the basket, so they must be in your purse." Here's what you get when you break this argument down into its premises and a conclusion:

Premises:

Either your keys are in your purse or they're in the basket.

Your keys aren't in the basket.

Conclusion:

Your keys are in your purse.

Here, the first premise sets up a pair of possibilities, and the second premise rules out one of these possibilities. So, the conclusion that the other possibility must be true completes a valid argument.

New York state of mind

Suppose that your friend Amy, describing where she lives, makes the following argument: "Manhattan is in New York. And Hell's Kitchen is in Manhattan. My apartment (which I inhabit) is in Hell's Kitchen, so I live in New York." Here's a cleaned-up version of this argument:

Premises:

I live in my apartment

My apartment is in Hell's Kitchen.

Hell's Kitchen is in Manhattan.

Manhattan is in New York.

Conclusion:

I live in New York.

Again, the structure of implication leads from the premises to the conclusion, so the argument is valid.

Thinking Again: Logical Fallacies

A *logical fallacy* is a flaw or underlying weakness in an argument that, on the surface, seems strong. In some cases, this flaw lies in the argument's logical structure. In other cases, it's hiding in one of the premises or in an unstated assumption.

In *Sophistical Refutations*, the sixth and final book of Aristotle's great treatise on logic, *Organon*, the philosopher laid the foundation for later study of logical fallacies. Medieval scholastics such as Boethius and Aquinas continued this study, and their collective influence continues today in textbooks, critical thinking curricula,

and debate training. (Chapter 2 offers more information about all these logic scholars.)

In this section, I introduce a variety of common logical fallacies by dividing them into categories that help you make sense of them.

Fallacies of relevance

Fallacies of relevance hinge on information that has no direct bearing on the issue being considered in the argument. Here are a few common fallacies of relevance.

Ad hominem

The phrase *ad hominem* is Latin for "to or against the person." An ad hominem argument undermines the person making the argument rather than the argument itself.

Example: Averey says that a snowstorm is approaching, but they're probably wrong because they're not a meteorologist.

Counterargument: What Averey does for a living isn't relevant to whether a snowstorm is, in fact, approaching. A better course of action would be to check a reliable source of weather data.

Straw man

A *straw man* is a weak argument that's vulnerable to attack, and a *straw man fallacy* is a misrepresentation of an argument to make it easier to attack.

Example: Failing to support this crime bill is essentially giving up our city streets to violent criminals, and nobody wants that.

Counterargument: Sure, nobody wants violence on the streets — *and* this crime bill isn't a good path to solving that problem.

Red herring

A *red herring* is an irrelevant consideration that diverts attention from an argument's core issue.

Example: Why worry about environmental issues when so many unemployed people exist?

Counterargument: We don't have to choose to address only one of these important issues. The best response may be to address both environmental issues *and* unemployment.

Appeal to emotion

An *appeal to emotion* is a logical fallacy in which the speaker attempts to persuade by triggering emotional reactions instead of presenting valid reasoning or evidence.

Example: We have to ban this video game. Think of the children!

Counterargument: Of course, we all care about the well-being of children. But what, exactly, is your concern about this video game?

Appeal to belief

An *appeal to belief* sidetracks evidence by focusing excessively on the other person's skepticism.

Example: You don't believe that extraterrestrial aliens exist because you don't want to believe it.

Counterargument: If you present me with a compelling argument that extraterrestrial aliens exist, you'll convince me. Until then, I'm not convinced.

Fallacies of weak induction

A *fallacy of weak induction* is a logical fallacy in which the premises offered in support of a conclusion aren't strong enough to justify that conclusion — even if they are relevant and seem reasonable. Unlike fallacies of relevance (which use distractions), weak induction fallacies offer relevant but insufficient support. Here are a few examples.

Correlation-causation fallacy

A *correlation-causation* fallacy occurs when a pair of related facts are misunderstood as a one fact causing the other.

Example: It's actually dangerous to go to the hospital when you're sick: More people die in hospitals than anywhere else.

Counterargument: Hospitals don't *cause* people to die. People in hospitals die with greater frequency *because* they're not well to begin with.

Appeal to authority

An *appeal to authority* assumes that something is true because an authority figure says it is.

Example: An online doctor says this diet works, so it must be effective.

Counterargument: Okay, but doctors can have differing opinions. What evidence does this online doctor cite to justify their claim?

Appeals to authority become even more shaky when the authority referenced has no relevant expertise.

Example: My favorite actor says this diet works, so it must be effective.

Counterargument: He may be your favorite actor, but what evidence does he cite to justify his claim?

Appeal to ignorance

An *appeal to ignorance* asserts that a proposition is true because it hasn't yet been proven false, or vice versa.

Example: Decades of research have failed to prove that ghosts don't exist, so there must be something to the idea that they exist.

Counterargument: If the research is inconclusive, then by definition we can't conclude anything one way or the other.

Hasty generalization

A *hasty generalization* makes a broad statement or stereotype based on a small sample size.

Example: Computer programmers are usually introverts — if you doubt this, I'll introduce you to the ones who work in my office.

Counterargument: The 20 or so programmers where you work may be quiet or shy, but the millions of programmers across the world aren't necessarily introverts.

Slothful induction

A *slothful induction* fallacy occurs when someone refuses to accept a well-supported conclusion despite strong evidence, typically by attributing the outcome to coincidence or denying the pattern.

Example: Even though I've crashed my car five times while texting, it's not because I was distracted. It was just bad luck.

Counterargument: Despite an obvious pattern, you're rejecting the reasonable conclusion that texting while driving is the cause of your accidents.

Gambler's fallacy

The *gambler's fallacy* is the belief that future probabilities are affected by past events in a random process.

Example: I've lost the last five times I played cards with my friends, so I'm due for a win!

Counterargument: The cards don't have a memory, so you're not due for a win or a loss. But if your friends play cards better than you do, then you're more likely due for another loss.

Fallacies of presumption

Fallacies of presumption occur when an argument assumes something that hasn't been proven or builds on unjustified premises. These fallacies sneak in assumptions that bias or distort the reasoning process. Here are a few common examples.

Excluded middle (all-or-nothing)

An *excluded middle* (or *all-or-nothing*) argument presents a pair of extremes as if they're the only possibilities when, in reality, more moderate alternatives do exist.

Example: If you can't be the best at what you do, don't bother doing it at all.

Counterargument: You can be good or even very good at something without necessarily being the best.

False dilemma

A *false dilemma* presents only two options when more than two exist or when the issue is more complicated than a simple choice.

Example: If you're not cheering for our team at Saturday's game, then you may as well be cheering for the other team.

Counterargument: I hope our team wins, and I'll be rooting for them, but I've got a job on Saturday that takes priority.

Cherry-picking (suppressed evidence)

Cherry-picking (also known as *suppressed evidence*) is ignoring or omitting relevant facts that would undermine the argument you're making.

Example: This medicine is perfectly safe — it passed a series of rigorous trials to get on the market.

Counterargument: But you're leaving out the important fact that the medicine also caused side effects in 40 percent of users.

Loaded question

A *loaded question* is a query that can't be easily answered because it contains a hidden assumption.

Example: Have you stopped cheating on your exams — yes or no?

Counterargument: I've never cheated on an exam.

Slippery slope

A *slippery slope* argument asserts that a relatively small first step will inevitably lead to a chain of related (negative) events.

Example: If we allow students to redo homework for a better grade, soon they'll expect to redo exams, and then they'll demand to write their own report cards.

Counterargument: We can stop this from occurring by allowing students to redo their homework, but explicitly not permitting them to redo exams or write their own report cards.

Begging the question

In common speech, the term *begging the question* is often loosely used to refer to a statement that causes a new question to be raised. Traditionally, though, begging the question more closely means "sidestepping the issue" — that is, glossing over a key consideration by assuming what you're trying to prove.

Example: Literacy is important because everybody needs to know how to read.

Counterargument: Hold on! You may be right, but *why* exactly does everybody need to know how to read?

False equivalence

A *false equivalence* is a logical fallacy in which two considerations are incorrectly presented as being morally, logically, or factually equivalent — when, in fact, they are not. The comparison overlooks critical differences.

Example: Everybody lies from time to time, so Mayor Smith shouldn't be called out for lying.

Counterargument: Mayor Smith's lies swindled taxpayers out of thousands of dollars. That's not just lying; it's fraud, and fraud is against the law.

Circular reasoning

An argument with *circular reasoning* begins with the conclusion it's claiming to prove.

Example: The information in my textbook must be correct because it's from a textbook, so it has to be right.

Counterargument: You can't use the textbook itself to verify information in the textbook. You need to find at least one other source of verification.

Fallacies of unwarranted assumptions

As with fallacies of presumption (which I discuss in the previous examples), *fallacies of unwarranted assumptions* rest on hidden statements that are assumed to be true.

The difference between these types of fallacies is subtle.

- **Fallacies of presumption** involve arguments that presume something that hasn't been proven, usually in a way that biases the reasoning or structure of the argument from the start.
- **Fallacies of unwarranted assumptions,** on the other hand, include any argument that takes for granted an unsupported assumption — whether about facts, causation, categories, or outcomes.

Here are a few common examples.

Post hoc, ergo propter hoc

The Latin phrase *post hoc, ergo propter hoc* means "after this, therefore because of this" — that is, assuming that because one event followed another, the later event was also caused by the earlier.

Example: I got sick after eating at that restaurant; the food must have been bad.

Counterargument: Possibly — or perhaps the food had nothing to do with it, and you would have gotten sick anyway.

Tu quoque (whataboutism)

In Latin, *tu quoque* means "you too." In this logical fallacy, someone's argument is dismissed because they are being hypocritical, not because it lacks reason or evidence.

Example: You say smoking is bad for my health, but you smoke too!

Counterargument: Just because I smoke, that doesn't prevent smoking from being an unhealthy habit.

Bandwagon fallacy

The *bandwagon fallacy* is the assumption that something is good or correct because it's popular — that is, because everyone is jumping on the bandwagon.

Example: Everyone I know is buying stock in this company, so it must be a good investment.

Counterargument: Just because a lot of people are buying a certain company's stock doesn't mean they won't all lose their money.

Middle-ground fallacy

A *middle-ground fallacy* assumes that the middle position between two extremes is necessarily correct.

Example: Jack said that Topeka is the third-largest city in Kansas, and Jill said it's the fifth-largest, so it must be the fourth-largest.

Counterargument: Either Jack or Jill, or both of them, is wrong, but the only way to know is to check a few reliable sources.

Fallacies of ambiguity

Fallacies of ambiguity occur when an argument uses unclear, vague, or misleading language — typically words or sentence structures that have multiple meanings — to confuse or mislead the audience.

Notice how each of the following fallacies exploits language that shifts meaning in subtle ways, often without the listener noticing.

Appeal to purity (no true Scotsman fallacy)

An *appeal to purity* is a logical fallacy in which someone redefines a group or category in order to exclude counterexamples and protect a generalization from criticism. It's sometimes called the *no true Scotsman fallacy*, but it works just as well in any country.

Example: Irv: "An artist doesn't need to rely on digital tools." James: "But I use digital tools, and so do a lot of artists." Irv: "That just proves that none of you are *real* artists, doesn't it?"

Counterargument: Lots of artists use digital tools, and you're free to dislike their art, but they're still artists.

Equivocation

Fallacies of *equivocation* use a word or phrase with multiple meanings in a misleading way.

Example: The senator says she supports free speech. But newspapers aren't free because you have to pay for them, so the senator is opposed to journalism.

Counterargument: Free speech means "unrestricted speech," not "speech delivered without cost." The senator's words say nothing about journalism.

Amphiboly

Fallacies of *amphiboly* arise from grammatical structure or imprecise wording that leads to misinterpretation.

Example: According to this article, researchers found fewer cases of heart disease in people who exercised regularly with dogs. So, if you want to stay heart-healthy, start exercising with your dog.

Counterargument: I think the article means that exercising and having a dog can reduce your risk of heart disease. It doesn't say anything about exercising with your dog.

Fallacy of accent

In a *fallacy of accent*, the meaning of a statement changes depending on which word or phrase is emphasized, often through selective quoting or a shift in tone.

Example: Film reviewer Chelsea Harper writes, "You may have heard that the new film by Angus Schlimm is a brilliant dystopian thriller, but if you did, you heard wrong." A press kit quotes Chelsea Harper as having written, ". . . the new film by Angus Schlimm is a brilliant dystopian thriller. . . ."

Counterargument: The press kit deliberately took some of Chelsea Harper's words out of context. Anybody who reads what she actually wrote would know that she disliked the film.

Fallacy of composition

A *fallacy of composition* assumes that what's true of the parts must be true of the whole.

Example: Water isn't really wet — it's made of hydrogen and oxygen, neither of which is wet.

Counterargument: The whole created by its parts can have emergent attributes — that is, features that are present in none of the parts.

Fallacy of division

On the flip side of a fallacy of composition (which I discuss in the previous example), a *fallacy of division* assumes that what's true of the whole must be true of the parts.

Example: My favorite team is the best in the league, so each member of my team is also the best at their position.

Counterargument: The parts of a whole don't necessarily share all the attributes of the whole: The weakest player on your favorite team may not be better than the strongest player in that position on another team.

Understanding What Logic Is and Isn't

Because logic is so loosely defined in common speech, people have all sorts of misconceptions about it. In some circles, logic is revered as the greatest human endeavor. In others, it's despised and looked upon as an academic exercise for elitists in the ivory tower, which is remote from the challenges that face ordinary people in their daily lives.

Logic may have both a good and a bad reputation because people think it's something it isn't. In this section, I examine what logic is and what it isn't.

You can use Table 3-1 as a handy reference to compare what logic can and can't do.

TABLE 3-1 The Cans and Cannots of Logic

Logic Can't	Logic Can
Create a valid argument.	Critique a given argument for validity.
Tell you what is true or false in reality.	Tell you how to work with true and false statements.
Tell you if an argument is sound.	Tell you if an argument is valid.
Justify conclusions arrived at by induction (generalization).	Justify conclusions arrived at by deduction (narrowing the possibilities).
Make an argument rhetorically stronger (more convincing).	Provide the basis for rhetorical improvement (making your arguments more persuasive).

Thinking versus logic

Reasoning, or even just plain thinking, is a hugely complex process that's only barely understood. And — to part company with Mr. Spock and his clan — logic is one piece of the reasoning process, but it doesn't make up the whole thing.

Think about the type of reasoning you can use to resolve a dispute between two children. You'd need to observe what is happening, connect the situation to past experiences, and try to figure out what may happen in the future. You may choose to act harshly by threatening to punish the squabbling children. You may act kindly and reassure them. You may remain impartial and listen to both sides of the story. Or you may take a little piece from all these approaches.

In short, you would have a lot of options, and you'd somehow find a way to keep the peace. (Or maybe none of your strategies would work, but that's a subject for *Parenting For Dummies,* not *Logic For Dummies.*) All of your options involve thinking. Even sending both kids to their rooms without any attempt to understand what's going on requires the kind of thinking that humans do best, but that even the smartest dog cannot do.

However, if you limited yourself to logic when keeping the peace between your kids, you wouldn't get anywhere. Logic doesn't tell you what to do or how to act — it's not that kind of a tool. So, when you respond to a situation by

observing, thinking, and not letting emotion sway your actions, you're calling on a whole host of abilities that are far more sophisticated (and also far less certain!) than logic.

REMEMBER

Don't get logic and thinking confused. Logic is just one aspect of thinking, but it's an aspect that has been studied so rigorously that it's understood very well. In this way, logic is a lot more — and also a lot less — than just clear thinking unclouded by emotion.

Reality — what a concept!

Because you're constantly working with true and false statements in logic, you may think the point of it is to tell you what's true and what's false — in other words, you may think that logic is supposed to tell you the nature of reality in objective terms. But, in fact, logic can't tell you what's objectively true or false — it can only tell you what's true or false relative to other statements you already know (or believe) to be true or false.

For example, look at these two statements:

> New Jersey is a U.S. state.
>
> Stealing is always wrong.

The first statement seems simple to verify as true, but the second seems difficult to verify at all. In either case, however, logic doesn't tell you right from wrong or even whether New Jersey is a state — you have to find other sources to confirm those things. What logic *does* tell you is whether an argument is valid — that is, whether a set of true premises produces a true conclusion.

REMEMBER

The only exception to this rule involves statements that are true in only a very limited and trivial sense. For example:

> All blue things are blue.
>
> Two plus two either equals four or doesn't equal four.
>
> If you're a salesman living in Denver, then you're a salesman.

Statements like these are called *tautologies*. Tautologies have an important role to play in logic, and you find out about that role in Chapter 6. For now, though, just understand that the main reason they're so reliably true is because they don't really convey any information about the world. As far as figuring out what in this crazy, complex world is true or false — you're on your own.

The sound of soundness

REMEMBER

A *sound argument* is simply a valid argument with factually true premises, making the conclusion factually true as well.

Soundness goes hand in hand with reality. But, because logic doesn't tell you whether a statement is true or false, it also doesn't tell you whether an argument is sound. Even a valid argument will produce a lousy conclusion if you give it a bunch of rotten premises. For example:

Premises:

Taylor Swift is on Earth.

When the Messiah is on Earth, the end times have arrived.

Taylor Swift is the Messiah.

Conclusion:

The end times have arrived.

Here, you have a logically valid argument. But is it sound or unsound? I personally have a hard time accepting the third premise as true, so I would have to say it's an unsound argument. And you may well agree. The point is this: Because logic won't take a side about whether a statement is true in reality, logic also won't help you decide whether an argument is sound.

REMEMBER

Make sure you see the difference between a valid argument and a sound argument. A valid argument contains a big if: *If* all the premises are true, then the conclusion must also be true. A sound argument is a valid argument with one further condition: The premises *really are true*, so, of course, the conclusion really is true as well.

TIP

Note that when you start off with an invalid argument, logic has a lot to say: If an argument is invalid, it's always unsound as well. Figure 3-1 shows a little tree structure that may help you understand arguments better.

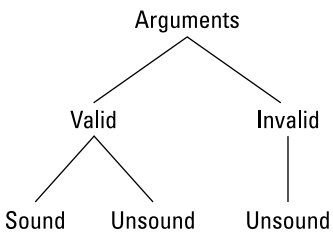


FIGURE 3-1: A look at a logic tree can help you tell where your arguments fall.

Deduction and induction

TIP

Because *deduction* begins with the same two letters as the word *decrease*, you can easily remember that in *deduction*, you start with a set of possibilities and *decrease* it until a smaller subset remains.

For example, a murder mystery is an exercise in deduction. Typically, the detective begins with a set of possible suspects — for example, the butler, the maid, the business partner, and the widow. By the end of the story, they have reduced this set to only one person — for instance,

> The victim died in the bathtub but was moved to the bed. Neither woman (the maid or widow) could have lifted the body, nor could the butler because of his war wound. Therefore, the business partner must have committed the crime.

Logic enables you to reason deductively with confidence. In fact, it's tailor-made for sifting through a body of factual statements (*premises*), ruling out plausible (believable) but inaccurate statements (*invalid conclusions*), and getting to the truth (*valid conclusions*). For this reason, logic and deduction are intimately connected.

Deduction works especially well in math, where the objects of study are clearly defined and where little or no gray area exists. For example, each of the counting numbers is either even or odd. So, if you want to prove that a number is odd, you can do so by ruling out that the number is divisible by 2.

TIP

Induction begins with the same two letters as the word *increase*, which can help you remember that in *induction*, you start with a limited number of observations and *increase* that number by generalizing.

For example, suppose you spend the weekend in a small town and the first five people you meet are friendly, so you inductively conclude the following: *Everybody here is so nice*. In other words, you started with a small set of examples, and you increased it to include a larger set.

WARNING

As apparently useful as induction is, it's logically flawed. Meeting five friendly people — or 10 or 10,000 — is no guarantee that the next one you meet won't be nasty. Meeting 10,000 people doesn't even guarantee that most people in the town are friendly — you may have simply met all the nice ones.

REMEMBER

Logic, however, is more than just a good, strong hunch that a conclusion is correct. The definition of logical validity demands that if your premises are true, the conclusion that naturally follows is also true. Because induction falls short of this standard, it's considered the great white elephant of both science and philosophy: It looks like it may work, but in the end it just takes up a lot of space in the living room.

THE SCIENCE OF INDUCTION

Induction is reasoning from a limited number of observations toward a general conclusion. A classic example: After observing that two or 10 or 1,000 ravens are black, you may decide that all ravens are black.

While inductive arguments may be convincing, they aren't logically valid. (You never know, a white or pink raven may be out there somewhere.) But, although induction is logically unreliable, scientists seem to use it all the time with great success to explain all sorts of events in the universe. The philosopher Karl Popper took on this apparent contradiction, showing that induction isn't necessary for scientific discovery.

Popper's answer, in a nutshell, is that science doesn't really use induction to prove anything. Instead, he said that scientists develop theories that fit their observations and then they *disprove* alternative theories. So, the last theory left standing becomes the accepted explanation until a better theory is found. (This may sound cavalier, as if any half-wit can come up with a crackpot theory and have it accepted. But scientists are extremely adept at disproving plausible theories, so don't quit your day job.)

Even though some thinkers still question Popper's proposition, many find it a good explanation that resolves a centuries-old philosophical problem.

See the "Whose Logic Is It, Anyway?" section later in this chapter for more examples of how and where logic is applied.

Rhetorical questions

Rhetoric is the study of what makes an argument convincing.

Take a look at the following argument:

Premises:

Science can't explain everything in nature.

Anything in nature can be explained either by science or by the existence of God.

Conclusion:

God exists.

Even though this may or may not be a sound argument, it's a valid argument. However, it may not be a *cogent* (clear, logical, and convincing) argument.

Operating on reasonable doubt

What makes an argument cogent is close to what makes it sound (see "The sound of soundness" earlier in this chapter). In a sound argument, the premises are true beyond all doubt. In a cogent argument, the premises are true beyond a reasonable doubt.

The phrase *reasonable doubt* may remind you of a courtroom. This connection makes sense because lawyers make logical arguments that are intended to convince a judge or a jury of a certain conclusion. In order to be cogent, the lawyer's arguments must be valid, and the premises of their arguments must be believable. (See the section "Tell it to the judge (law)," where I talk about the relevance of logic in a legal context.)

Considering that logic is concerned with validity rather than soundness, how does logic enter into the process of making an argument cogent? It does so by drawing a clear distinction between the underlying form of an argument and its content.

REMEMBER

The form of an argument is what logic is all about. If the form doesn't function correctly, your argument falls apart. However, when it does function correctly, you can move on with confidence to the content of your argument.

Evaluating a cogent argument

Take a look at the argument at the beginning of the section "Rhetorical questions" earlier in the chapter. I say that this argument is valid, but it may not be cogent. Why? For one thing, the second premise (*Anything in nature can be explained either by science or by the existence of God*) is full of holes. Consider the following:

- What if science can't currently explain everything in nature but someday is able to?
- What happens if something other than science or the existence of God can explain some things?
- What if there's simply no explanation for everything you see in nature?

Each of these questions is a plausible exception to the argument. You need to address these exceptions if you want your argument to convince intelligent, clear-thinking people that your reasoning is correct. In other words, you need to address the rhetorical issues within your argument.

In reality, however, appealing to the intelligence of an audience may be less effective than an appeal to powerful emotions and irrational beliefs. In such cases, a simple, flawed argument may be more convincing than a sound argument that the audience has trouble understanding. A shaky argument from a speaker whom the audience likes can be convincing where a brilliant but obnoxious display of intellect may just be alienating.

REMEMBER

The study of what makes an argument cogent or convincing is useful and provocative, but it's also outside the scope of this book. From a purely logical standpoint, once an argument is valid, no further improvement is possible.

Whose Logic Is It, Anyway?

With all the restrictions placed upon it, you may think logic is too narrow to be of much use. But its narrowness is logic's great strength. Logic is like a laser — a tool whose best use is not illumination, but rather focus. A laser may not provide light for your home, but, like logic, its great power resides in its precision. The following sections describe just a few areas in which logic is commonly used.

Pick a number (math)

Mathematics is perfectly suited to use logic in all its power. In fact, logic is one of the three theoretical legs that math stands on. (The other two are set theory and number theory, if you're wondering.)

REMEMBER

Logic and math work well together because they're both independent from reality and because they're tools that are used to help people make sense of the world. For example, reality may contain three apples or four bananas, but the ideas of *three* and *four* are abstractions, even though they're abstractions that most people take for granted.

Math is made completely of such abstractions. When these abstractions get complicated — at the level of algebra, calculus, and beyond — logic can be called on to help bring order to their complexities. Mathematical ideas such as number, sum, fraction, and so on are clearly defined without exceptions. That's why statements about these ideas are much easier to verify than a statement about reality, such as *People are generally good at heart* or even *All ravens are black.*

Fly me to the moon (science)

Science uses logic to great advantage. Like math, science uses abstractions to make sense of reality and then applies logic to these abstractions.

Scientists attempt to understand reality by

- Reducing reality to a set of abstractions, called a *model*
- Working within this model to reach a conclusion
- Applying this conclusion to reality again

Logic is instrumental during the second step, and the conclusions that science attains are, not surprisingly, logical conclusions. This process is most successful when a good correlation exists between the model and reality and when the model lends itself well to the type of calculations that logic handles comfortably.

The areas of science that rely most heavily on logic and math are the *quantifiable sciences*, such as physics, engineering, and chemistry. The *qualitative sciences* — biology, physiology, and medicine — use logic but with a bit less certainty. Finally, the *social sciences* — such as psychology, sociology, and economics — are the sciences whose models bear the least direct correlation to reality, which means they tend to rely less on pure logic.

Boot up or power down (computer science)

Medicine used to be called the youngest science, but that title has been handed over to computer science. A huge part of the success of the computer revolution rests firmly on logic.

Every action your computer completes happens because of a complex structure of logical instructions. At the *hardware* level — the physical structure of the machine — logic is instrumental in the design of complex circuits that make the computer possible. And, at the *software* level — the programs that make computers useful — computer languages based on logic provide for the endless versatility that sets the computer apart from all other machines.

See Chapter 20 for a deeper discussion of logic as it relates to computers.

Tell it to the judge (law)

REMEMBER

As with mathematics, laws exist primarily as sets of definitions: contracts, torts, felonies, intent to cause bodily harm, and so on. These legal concepts all come into being on paper and then are applied to specific cases and interpreted in the courts. A legal definition provides the basis for a legal argument, which is similar to a logical argument.

For example, to demonstrate copyright infringement, a plaintiff may need to show that the defendant published a certain quantity of material under their own name, for monetary or other compensation, when this material was protected by a preexisting copyright.

These criteria are similar to the premises in a logical argument: If the premises are found to be true, the conclusion — that the defendant has committed copyright infringement — must also be true.

Find the meaning of life (philosophy)

Logic had its birth in philosophy and is often still taught as an offshoot of philosophy rather than math. Aristotle invented logic as a method for comprehending the underlying structure of reason, which he saw as the motor that propelled human attempts to understand the universe in the widest possible terms.

REMEMBER

As with science, philosophy relies on models of reality to help provide explanations for what people see. Because the models are rarely mathematical, however, philosophy tends to lean more toward rhetorical logic than mathematical logic.

2 Flourishing with Formal Sentential Logic

IN THIS PART . . .

Find out how to translate statements from English into sentential logic (SL) by using constants, variables, and the five logical operators.

See how to decide whether an SL statement is true or false.

Discover truth tables, which are powerful tools for detecting the truth value of SL statements.

Use quick tables as a shortcut to solve problems in logic.

Understand how truth trees have all the advantages of truth tables and quick tables but none of the disadvantages.

IN THIS CHAPTER

- Introducing formal logic
- Defining the five logical operators
- Recognizing how logic is similar to arithmetic
- Translating statements from SL to English, and vice versa

Chapter 4

Engaging with Formal Affairs

If you look at a few logical arguments, which you can do in Chapter 3, you may get the sneaking suspicion that they all have a lot in common. And you'd be right. Over the centuries, logicians have examined their fair share of argument examples, and they've found that certain patterns of arguments come up again and again. They realized that they could capture these patterns by using a small number of symbols, and then analyze the patterns for their common features.

In this chapter, I introduce *formal logic*, a foolproof set of methods for determining whether an argument is valid or invalid. I show you how to represent statements with placeholders called *constants* and *variables*, and I discuss the five *logical operators* used for connecting simple statements to build more complex ones.

The logical operators work very much like the familiar operators in arithmetic (addition, subtraction, and so on). I point out their similarities so you can get comfortable with the new symbols. And I show you how to translate statements in English into logical statements and vice versa.

Observing the Formalities of Sentential Logic

REMEMBER

As I mention in Chapter 2, *sentential logic*, or SL, also known as *propositional logic*, is one of two forms of classical formal logic. (The other form is *quantification logic*, or QL, also known as *predicate logic* or *first-order logic.*) I introduce SL in this chapter and dig deeper into it throughout Parts 2 and 3. I cover QL in Part 4.

You can communicate logical arguments in language, but natural languages — such as English — tend to be imprecise. Words often have more than one meaning and sentences can be misinterpreted.

To help solve this problem, mathematicians and philosophers developed sentential logic, a language specifically designed to express logical arguments with precision and clarity. Because SL is a symbolic language, it has the added advantage of allowing for calculation according to precisely defined rules and formulas. Just like with mathematics, as long as you follow the rules correctly, you can guarantee a correct answer.

In the following sections, I introduce a few types of symbols that SL uses to achieve the goals of precision and clarity.

Turning language into statement constants

If you ever spent a day in an algebra or pre-algebra class, you were probably exposed to that elusive little fellow known as *x*. Your teacher probably told you that *x* stood for a secret number and it was your job to make *x* talk. They then showed you all sorts of sadistic ways of torturing poor little *x* until at last it broke down and revealed its true numerical identity. Oh, what fun that was.

Making letters stand for numbers is one thing mathematicians are really good at doing. So it isn't surprising that formal logic, which was developed by mathematicians, also relies on letters as stand-ins. In the chapter introduction, I let on that logic uses *statements* rather than numbers, so it's logical to guess that in formal logic, letters stand for statements. For example:

> Let *K* = Katy is feeding her fish.
>
> Let *F* = The fish are joyfully flapping their little fins.

REMEMBER

When a letter stands for a statement in a language such as English, the letter becomes a *statement constant* in logic. By convention, constants are signified by capital letters.

When it comes to constants, logicians tend to like the capital letters *P* and *Q* the most. This is because *P* is the first letter in the word *proposition*, which means the same thing as statement in logic, and *Q* being next alphabetically just came along for the ride.

Having a truth value for every statement

As I tell you in Chapter 3, every statement in logic has a *truth value* that's either true or false. In formal logic, *true* is shortened to **T** and *false* to **F**.

For example, consider the truth values of these two statement constants:

Let *N* = The Nile is the longest river in Africa.

Let *L* = Leonardo DiCaprio is the king of the world.

As it happens, it's true that the Nile is the longest river in Africa, so the truth value of *N* is **T**. And it so happens that Leonardo DiCaprio is (still!) *not* the king of the world, so the truth value of *L* is **F**.

Boolean algebra, the precursor to formal logic, uses the value *1* to represent **T** and *0* to represent **F**. And computer logic also uses these two values. (I discuss Boolean algebra in Chapter 14 and computer logic in Chapter 20.)

Deploying the Five SL Operators

SL has five basic operators, as depicted in Table 4-1. These *logical operators* are similar to the arithmetic operators in that they take values you give them and produce a new value. However, logical operators really only deal with two values: the truth values, **T** and **F**. In the sections that follow, I explain each of the operators in Table 4-1.

TABLE 4-1 The Five Logical Operators

Operator	Technical Name	What It Means	Example
~	Negation operator	Not	$\sim x$
&	Conjunction operator	And	$x \,\&\, y$
$\vee$	Disjunction operator	Or	$x \vee y$
$\rightarrow$	Conditional operator	If-then	$x \rightarrow y$
$\leftrightarrow$	Biconditional operator	If and only if	$x \leftrightarrow y$

In the table, I use the lowercase variables *x* and *y* to stand for overall patterns in SL, in contrast to uppercase statement constants *P* and *Q*, which I use for specific SL statements. Thus, when any SL statement that *x* stands for is true, then ~*x* is false. On the other hand, when a statement that *x* stands for is false, then ~*x* is true.

Feeling negative

REMEMBER

In English, you can turn any statement into its opposite by adding the word *not* and, perhaps, changing a few additional words. This is called *negating* the statement. Of course, negating a true statement turns it into a false statement, and negating a false statement makes it true. In general, then, every statement has the opposite truth value from its negation.

In SL, the negation operator, or not-operator, is a *tilde* (~). For example, I can tweak statement *N* by simply inserting one little word:

N = The Nile is the longest river in Africa.

~*N* = The Nile is *not* the longest river in Africa.

The addition of *not* transforms the original statement into its opposite, which is the negation of that statement. Having established that the value of *N* is **T** (see the section "Having a truth value for every statement" earlier in the chapter), I can conclude that the value of its negation is **F**.

REMEMBER

Other books about logic may use a dash (–), or another symbol that looks like a sideways L, for the not-operator rather than the tilde. I'm a tilde sort of guy, but whatever the symbol, it means the same thing.

Consider another example of negation:

L = Leonardo DiCaprio is the king of the world.

~*L* = Leonardo DiCaprio is *not* the king of the world.

In this case, after establishing that the value of *L* is **F** (see the section "Having a truth value for every statement" earlier in the chapter), you also know that that value of ~*L* is **T**.

This information is simple to summarize in a table, as follows:

x	~*x*
T	F
F	T

TIP

Memorize the information in this table. You'll be using it a lot in other chapters.

Feeling even more negative

Although negation is just the beginning, SL's little system of symbols has already grown more powerful than it looks. For example, given that the value of a new statement *R* is **T** and that its negation ~*R* is **F**, what can you say about the double negative statement ~~*R*?

If you guessed that the value of ~~*R* is **T**, give yourself a pat on the back. And as you do, notice that you were able to figure this out even though I didn't define *R*.

Witness the magic and power of logic. With just a few simple rules, you know that *any* true statement when negated twice *must* be true, even if you don't know exactly what the statement is. This guarantee is similar to the idea that as soon as you know that 2 apples + 3 apples = 5 apples, you know that your result (5) will be true no matter what you're counting: apples, dinosaurs, leprechauns — whatever.

Putting everything on the table

Because *R* can take on only two possible truth values — **T** or **F** — you can organize the information about ~*R*, ~~*R*, and so forth in the following table:

R	*~R*	*~~R*	*~~~R*
T	F	T	F
F	T	F	T

REMEMBER

Tables of this kind are called *truth tables.* The first column contains the two possible truth values of *R*: **T** and **F**. The remaining columns give the corresponding values for the different related statements: ~*R*, ~~*R*, and so on.

Reading truth tables is pretty straightforward. Using the sample truth table, if you know that the value of *R* is **F** and want to find out the value of ~~~*R*, you find the point where the bottom row crosses the last column. The truth value in this position tells you that when *R* is **F**, ~~~*R* is **T**.

Chapter 6 shows you how truth tables are a powerful tool in logic, but for now, I just use them to organize information clearly.

Displaying a show of ands

The symbol & is called the *conjunction operator* or, more simply, the *and-operator.* You can think of it as just the word *and* placed between two statements, joining them together to make a new statement.

Check out this statement:

> Albany is the capital of New York *and* Tom Brady was the quarterback for the New England Patriots.

Is this statement true or false? To decide, you need to recognize that it contains two smaller statements: one about Albany and another about Tom Brady. Its truth value depends on the truth value of both parts.

Because both parts are true, the statement as a whole is true. Suppose, however, that one of the statements is false. Imagine some alternate universe where Albany is not the capital of New York or where Tom Brady was never the quarterback of the Patriots. In either case, the truth value of the statement would be false.

TIP

Other books about logic may use a dot (·) or an inverted V for the and-operator rather than the ampersand (&). In still other books, x & y is simply written xy. Whatever the convention, the statements mean the same thing. For me, nothing says *and* like an ampersand.

In logic, you handle statements that involve the word *and* in a special way. First, each smaller statement is assigned a constant:

> Let A = Albany is the capital of New York.
>
> Let B = Tom Brady was the quarterback for the New England Patriots.

Then, you connect the two constants as follows:

$$A \,\&\, B$$

The truth value of this new statement is based on the truth values of the two parts you've joined together. If *both* parts are true, then the whole statement is true. On the other hand, if *either* part (or both) is false, then the whole statement is false.

For the and-operator, you can organize the truth values of *x* and *y* into a table:

x	*y*	*x* & *y*
T	T	T
T	F	F
F	T	F
F	F	F

TIP

Memorize the information in this table. Here's a quick way to remember it: An and-statement is true only when both parts of the statement are true. Otherwise, it's false.

Notice that the table for the and-operator has four rows rather than just the two rows I used for the not-operator (see the "Feeling negative" section earlier in the chapter). The tables are different because the and-operator always operates on two variables, so its table has to track all four paired values for *x* and *y*.

Digging for or

A statement may also be made up of two smaller statements joined together with the word *or*. Logic provides an operator for the word *or:* The *disjunction operator*, or just plain *or-operator*, is $\vee$.

Take a look at this statement:

> Albany is the capital of New York *or* Tom Brady was the quarterback for the New England Patriots.

If you designate the first part of the statement as A and the second part as B, you can connect the two constants A and B as follows:

$$A \vee B$$

Is this statement true? Just as with an and-statement, when both parts of an or-statement are true, the entire statement is true. Therefore, the statement $A \vee B$ has a truth value of **T**. However, with an or-statement, even if only one part is true, the statement as a whole is still true. For example:

> Let A = Albany is the capital of New York.
>
> Let S = Tom Brady was the shortstop for the Boston Red Sox.

Now the statement $A \vee S$ means:

> Albany is the capital of New York *or* Tom Brady was the shortstop for the Boston Red Sox.

Even though the second part of this statement is false, the whole statement is true because one part of it is true. Therefore, $A \vee S$ has a truth value of **T**.

But when *both* parts of an or-statement are false, the whole statement is false. For example:

> Let *Z* = Albany is the capital of New Zealand.
>
> Let *S* = Tom Brady was the shortstop for the Boston Red Sox.

Now the statement $Z \vee S$ means:

> Albany is the capital of New Zealand *or* Tom Brady was the shortstop for the Boston Red Sox.

This is a false statement because both parts of it are false. So $Z \vee S$ has a truth value of **F**.

For the or-operator, you can make a table with four rows that covers all possible combinations of truth values for *x* and *y*:

x	*y*	*x* ∨ *y*
T	T	T
T	F	T
F	T	T
F	F	F

TIP

Memorize the information in this table. Here's a quick way to remember it: An or-statement is false only when both parts are false. Otherwise, it's true.

In English, the word *or* has two distinct meanings:

- **Inclusive *or*:** When *or* means "this choice *or* that choice, *or both*," the possibility that both parts of the statement are true is *included.* An example of an inclusive *or* is when a parent says, "Before you go out, you need to clean your room *or* do your homework." Clearly, they mean for their child to do one of these tasks *or both of them.*
- **Exclusive *or*:** When *or* means "this choice *or* that choice, *but not both*," the possibility that both parts of the statement are true is *excluded.* An example of an exclusive *or* is when a parent says, "I'll give you money for pizza today *or* horseback riding tomorrow." They mean that their child gets money for one of these activities, *but not both.*

REMEMBER

English can be unclear, but logic isn't. By convention, in logic, the or-operator is always *inclusive.* If both parts of an or-statement are true, the statement also is true.

TECHNICAL STUFF

Both the inclusive and exclusive *or* are used in the design of logic gates, an integral part of computer hardware. (See Chapter 20 for more information about computer logic.)

Getting iffy

The symbol $\rightarrow$ is called the *conditional operator*, also known as the *if . . . then-operator* or just the *if-operator.* To understand how the if-operator works, take a look at this statement:

> If a wig is hanging off a bedpost in the guest room, then Aunt Doris is visiting.

You can see that the statement contains two separate statements, and you can represent each of them by a statement constant:

> Let W = A wig is hanging off a bedpost in the guest room.
>
> Let D = Aunt Doris is visiting.

Connect the two with the new if-operator:

$$W \rightarrow D$$

As with the other operators covered in this chapter, for the if-operator, you can make a table with four rows that covers all possible combinations of truth values for *x* and *y*:

x	*y*	$x \rightarrow y$
T	T	T
T	F	F
F	T	T
F	F	T

TIP

Memorize the information in this table. A quick way to remember it is that an if-statement is false only when the first part of it is true and the second part is false. Otherwise, it's true.

Other books on logic may use a symbol that looks like a U on its side — $\subset$ — for the if-operator rather than an arrow. Whatever the symbol, it means the same thing.

REMEMBER

There's nothing random about the way the if-operator looks. The arrow points from left to right for an important reason: When an if-statement is true and the first part of it is true, the second part *must be true as well.*

To make this relationship between if-statement parts clear, I need a couple new constants:

Let *B* = You're in Boston.

Let *M* = You're in Massachusetts.

Now, consider the statement

$$B \rightarrow M$$

This statement means "If you're in Boston, then you're in Massachusetts." Clearly, the statement is true, but why? Because Boston is completely inside Massachusetts.

The converse of a statement

When you reverse an if-statement, you produce a new statement called the *converse* of that statement. For example, here's an if-statement followed by its converse:

If-statement: If you're in Boston, then you're in Massachusetts.

Converse: If you're in Massachusetts, then you're in Boston.

WARNING

When an if-statement is true, it doesn't necessarily follow that its converse is true as well. Although the preceding if-statement is true, the converse is false. You may be in Concord, Provincetown, or any number of other places in Massachusetts.

The inverse of a statement

When you negate both parts of an if-statement, you get another statement called the *inverse* of that statement. For example, compare the following:

If-statement: If you're in Boston, then you're in Massachusetts.

Inverse: If you're not in Boston, then you're not in Massachusetts.

WARNING

When an if-statement is true, it doesn't necessarily follow that its inverse is true as well. Using the preceding example, even if you're not in Boston, you may still be somewhere else in Massachusetts.

The contrapositive of a statement

When you both reverse the order and negate the two parts of an if-statement, you get the *contrapositive* of the original statement. I know — it's getting deep. But here's an example:

If-statement: If you're in Boston, then you're in Massachusetts.

Contrapositive: If you're not in Massachusetts, then you're not in Boston.

REMEMBER

An if-statement and its contrapositive always have the same truth value. Using the example, if the first part is true — you're not in Massachusetts — it's obvious that you can't be in Boston.

TECHNICAL STUFF

Although a statement and its contrapositive always have the same truth value, in practice, proving the contrapositive of a statement is sometimes easier than proving the statement itself. (For more on how to construct proofs in SL, flip to Part 3.) Additionally, the converse of a statement always has the same truth value as the inverse of the same statement. This is because the converse and inverse are actually contrapositives of each other.

Getting even iffier

In SL, the if-and-only-if-operator ($\leftrightarrow$) is similar to the if-operator (see "Getting iffy" earlier in the chapter) but has more going on. The best way to understand the if-and-only-if-operator is to first establish an if-statement and then work your way along.

Consider this if-statement, introduced in "Getting iffy" earlier in this chapter:

> *If* a wig is hanging off a bedpost in the guest room, *then* Aunt Doris is visiting.

This statement says:

1. If you see a wig, then you *know* that Aunt Doris is here, *but*
2. If you see Aunt Doris, then you *can't be sure* that there is a wig.

This if-statement can be represented in SL as $W \rightarrow D$, with the arrow pointing in the direction of implication: Wig *implies* Doris.

Now consider this statement:

> A wig is hanging off a bedpost in the guest room *if and only if* Aunt Doris is visiting.

This statement is similar to the previous one, but it extends things a bit further. In this case, the statement says:

1. If you see a wig, then you *know* that Aunt Doris is here, *and*
2. If you see Aunt Doris, then you *know* that there is a wig.

This statement can be represented in SL as $W \leftrightarrow D$, with the double arrow providing a clue to its meanings: *both* wig implies Doris, *and* Doris implies wig.

Because the if-operator ($\rightarrow$) is also called the conditional operator, the if-and-only-if-operator ($\leftrightarrow$) is quite logically called the *biconditional operator.* A shorter way of referring to it is the *iff-operator.* But this stuff can be tricky enough, so to avoid confusion, I spell it out as *if-and-only-if*.

WARNING

Don't confuse the if-and-only-if-operator ($\leftrightarrow$) with the if-operator ($\rightarrow$).

TIP

Other books about logic may use $\equiv$ for the if-and-only-if-operator rather than the double arrow. Whatever the symbol, it means the same thing.

As with the other operators, for the if-and-only-if-operator, you can make a table with four rows that covers all possible combinations of truth values for *x* and *y*:

x	*y*	$x \leftrightarrow y$
T	T	T
T	F	F
F	T	F
F	F	T

TIP

Memorize the information in this table. Here's a quick way to remember it: An if-and-only-if-statement is true only when both parts of it have the same truth value. Otherwise, it's false.

An important feature of the if-and-only-if-statement is that both parts of the statement are *logically equivalent*, which means that one can't be true without the other.

Check out two more examples of if-and-only-if-statements:

> You're in Boston if and only if you're in Beantown.
>
> A number is even if and only if you can divide it by two without a remainder.

The first statement is saying that Boston *is* Beantown. The second statement is pointing out the equivalence of its two parts — being an even number is equivalent to being evenly divisible by two.

Seeing How SL Is Like Simple Arithmetic

As I discuss in the section "Deploying the Five SL Operators" earlier in this chapter, SL is a close cousin to math in that the operators for both disciplines take the values you give them, operate on those values, and produce a new value. But the similarity doesn't stop there. After you see a few other similarities, SL becomes much easier to understand.

The ins and outs of values

In arithmetic, each of the four basic operators turns two numbers into one number. For example:

$$6 + 2 = \mathbf{8} \qquad 6 - 2 = \mathbf{4} \qquad 6 \times 2 = \mathbf{12} \qquad 6 \div 2 = \mathbf{3}$$

REMEMBER

The two numbers you start out with are called *input values*, and the number you end up with is the *output value*.

Naming operators by the input values

In each case, placing the operator between two input values (6 and 2) produces an output value (in boldface). When you have two input values, the operators are called *binary operators.*

The minus sign (–) serves another purpose in math. When you place it in front of a positive number, the minus sign turns that number into a negative number. But when you place it in front of a negative number, the minus sign turns that number into a positive number. For example:

$$- \; -4 = 4$$

In this case, the first minus sign operates on one input value (–4) and produces an output value (4). When used in this way, the minus sign is a *unary operator*, because you have only one input value.

In arithmetic, you have to worry about an infinite number of values. SL, however, has only two values: **T** and **F**. (For more about these truth values, jump back to the "Having a truth value for every statement" section earlier in this chapter.)

Comparing math operators and logic operators

As with arithmetic, logic has four binary operators and one unary operator. In SL, the binary operators are &, $\vee$, $\rightarrow$, and $\leftrightarrow$, and the unary operator is ~. (Each of these operators is covered in the section "Deploying the Five SL Operators" earlier in this chapter.)

REMEMBER

For both types of operators, the same basic rules apply in SL as in arithmetic:

- Place a binary operator between any pair of input values and you get an output value.
- Place a unary operator in front of an input value and you get an output value.

For example, starting with a pair of input values, **F** and **T**, in that order, you can combine them by using the four binary operators as follows:

$$\mathbf{F} \,\&\, \mathbf{T} = \mathbf{F} \qquad \mathbf{F} \vee \mathbf{T} = \mathbf{T} \qquad \mathbf{F} \rightarrow \mathbf{T} = \mathbf{T} \qquad \mathbf{F} \leftrightarrow \mathbf{T} = \mathbf{F}$$

In each case, the operator produces an output value, which of course is either **T** or **F**. Similarly, placing the unary operator ~ in front of either input value **T** or **F** produces an output value:

$$\sim\mathbf{F} = \mathbf{T} \qquad \sim\mathbf{T} = \mathbf{F}$$

There's no substitute for substitution

Even if you've had only the slightest exposure to algebra, you know that letters can stand for numbers. For example, if I tell you that

$$a = 9 \text{ and } b = 3,$$

you can figure out that

$$a + b = 12 \qquad a - b = 6 \qquad a \times b = 27 \qquad a \div b = 3$$

When you're working with constants in SL, the same rules apply. You just need to substitute the correct values (**T** or **F**) for each constant. For example, look at the following problem:

Given that *P* is true, *Q* is false, and *R* is true, find the values of the following statements:

1. $P \vee Q$

Substitute **T** for *P* and **F** for *Q* to get $\mathbf{T} \vee \mathbf{F}$, which equals **T**.

2. $P \rightarrow R$

Substitute **T** for *P* and **T** for *R* to get $\mathbf{T} \rightarrow \mathbf{T}$, which equals **T**.

3. $Q \leftrightarrow R$

Substitute **F** for *Q* and **T** for *R* to get $\mathbf{F} \leftrightarrow \mathbf{T}$, which equals **F**.

Parenthetical guidance suggested

In arithmetic, parentheses are used to group numbers and operations together. For example:

$$-((4 + 8) \div 3)$$

In this expression, the parentheses tell you to solve $4 + 8$ first, which gives you 12. Then, moving outward to the next set of parentheses, you solve $12 \div 3$, giving you 4. Finally, the unary negation operator $(-)$ changes this to -4.

In general, then, you start with the innermost pair of parentheses and work your way out. SL uses parentheses in the same way. Using the example from the previous section, given that P is true, Q is false, and R is true, find the value of the following statement:

$$\sim((P \vee Q) \rightarrow \sim R)$$

1. Start with the innermost pair of parentheses, and $P \vee Q$ becomes **T** $\vee$ **F**, which simplifies to **T**.
2. Move outward to the next pair of parentheses, **T** $\rightarrow \sim R$ becomes **T** $\rightarrow$ **F**, which simplifies to **F**.
3. Apply the negation (~) that appears outside all the parentheses, which changes this **F** to **T**.

REMEMBER

The process of reducing a statement with more than one value down to a single value is called *evaluating the statement*. It's an important tool that you find out more about in Chapter 5.

(Not) Getting Lost in Translation

SL is a language, so after you know the rules, you can translate back and forth between SL and English — or Spanish or Chinese, for that matter — but I'll stick to English in this book! (However, if you're into Spanish or Chinese, check out *Spanish For Dummies* or *Chinese For Dummies*.)

REMEMBER

The main strength of SL is that it's clear and unambiguous. These qualities make translating an SL statement into English easy. That's why I call this translation direction the *easy way*. English, however, can be unclear and ambiguous. (See the section "Digging for or" earlier in the chapter for evidence that even the simple word *or* has several meanings, depending on how you use it.) Because you have to be very careful when translating sentences from English into SL, I call this translation direction the *not-so-easy way*.

TIP

Discovering how to translate statements in both directions will sharpen your understanding of SL by making the concepts behind all these odd little symbols a lot clearer. And as you navigate the other chapters in Part 2, if you start to get confused, just remember that every statement in SL, no matter how complex, can also be stated in English.

The easy way — translating from SL to English

Sometimes examples offer the best explanation, just like a picture is worth a thousand words. So, in the following sections, I give you examples of several ways to translate each type of operator. They're all pretty straightforward, so pick your favorite and run with it.

Throughout this section, I use the following statement constants:

Let *A* = Aaron loves Alma.

Let *B* = The boat is in the bay.

Let *C* = Cathy is catching catfish.

Translating statements with ~

You can translate the statement ~*A* into English in any of the following ways:

It is not the case that Aaron loves Alma.

It isn't true that Aaron loves Alma.

Aaron *does not love* Alma.

Aaron *doesn't love* Alma.

Translating statements with &

Here are two ways to translate the SL statement A & B into English*:*

Aaron loves Alma *and* the boat is in the bay.

Both Aaron loves Alma *and* the boat is in the bay.

Translating statements with ∨

Here are two ways to translate $A \vee C$ into English*:*

Aaron loves Alma *or* Cathy is catching catfish.

Either Aaron loves Alma *or* Cathy is catching catfish.

Translating statements with $\rightarrow$

You can translate the statement $B \rightarrow C$ into English in any of the following ways:

> *If* the boat is in the bay, *then* Cathy is catching catfish.
>
> The boat is in the bay *implies* Cathy is catching catfish.
>
> The boat is in the bay *implies that* Cathy is catching catfish.
>
> The boat is in the bay *only if* Cathy is catching catfish.

Translating statements with $\leftrightarrow$

There's really only one clear way that I know to translate the SL statement $C \leftrightarrow B$ into English:

> Cathy is catching catfish *if and only if* the boat is in the bay.

Translating more complex statements

For more complex statements, you can refer to the guidelines I discuss earlier in the chapter, in the section "Seeing How SL Is Like Simple Arithmetic." You simply translate statements step-by-step, starting from inside the parentheses. For example:

$$(\sim A \,\&\, B) \vee \sim C$$

The part inside the parentheses is $\sim A \,\&\, B$, which translates into English as

> Aaron doesn't love Alma and the boat is in the bay.

Adding on the last part of the statement produces:

> Aaron doesn't love Alma and the boat is in the bay *or* Cathy is not catching catfish.

Notice that although the sentence is technically correct, it's somewhat confusing because the parentheses are gone and everything runs together. A good way to clean it up is to translate it into English as

> *Either* Aaron doesn't love Alma and the boat is in the bay *or* Cathy is not catching catfish.

The word *either* clarifies just how much the word *or* is meant to encompass. In contrast, the SL statement

$$\sim A \,\&\, (B \vee \sim C)$$

can be translated into English as

> Aaron doesn't love Alma and *either* the boat is in the bay *or* Cathy is not catching catfish.

Now take a look at another example:

$$\sim(A \rightarrow (\sim B \,\&\, C))$$

Starting from the inner parentheses, you translate $\sim B \,\&\, C$ as

> The boat is not in the bay and Cathy is catching catfish.

Moving to the outer parentheses, $A \rightarrow (\sim B \,\&\, C)$ becomes this in English:

> *If Aaron loves Alma, then both* the boat is not in the bay and Cathy is catching catfish.

Note the addition of the word *both* to make it clear that the original and- statement is in parentheses. Finally, add on the ~ to get

> *It isn't the case that* if Aaron loves Alma, then both the boat is not in the bay and Cathy is catching catfish.

Okay, so it's a little complicated, but it makes about as much sense as a sentence like that ever will. Outside a class in formal logic, you'll probably never have to translate statements much more complicated than these. In any case, reflect for a moment on the fact that SL can handle statements of any length with perfect clarity in a way that English struggles to achieve .

The not-so-easy way — translating from English to SL

Each of the four binary operators in SL (&, $\vee$, $\rightarrow$, and $\leftrightarrow$) is a connector that joins a pair of statements together. In English, words that connect statements together are called *conjunctions.* Here are some examples of English conjunctions:

although	if . . . then	or
and	either . . . or	so
but	neither . . . nor	therefore
however	nevertheless	though
if and only if	only if	yet

REMEMBER

The not-operator (~) is usually translated into English as *not*, but it can appear in other clever disguises, such as *contractions* (can't, don't, won't, and so on). With all this variety in language, an exhaustive account of how to turn English into SL would probably take ten books this size. What follows, then, is a slightly shorter explanation. In this section, I list some of the most common words and phrases, discuss each one, and then provide an example of how to translate it into SL.

First, though, I need to define some constants:

Let *K* = Chloe lives in Kentucky.

Let *L* = Chloe lives in Louisville.

Let *M* = I like Mona.

Let *N* = I like Nunu.

Let *O* = Chloe likes Olivia.

But, though, however, although, nevertheless, yet . . .

Lots of words in English (as listed in the preceding heading) link statements together with the same logical meaning as the word *and.* Here are a few examples:

I like Mona, *but* I like Nunu.

Though I like Mona, I like Nunu.

I like Mona; *however*, I like Nunu.

Although I like Mona, I like Nunu.

I like Mona; *nevertheless*, I like Nunu.

I like Mona, *yet* I like Nunu.

Each of these words provides a slightly different meaning, but because you're just talking about logic here, you translate them all into SL as

$M \,\&\, N$

REMEMBER

After you translate a statement from English to SL, the rules of SL take over to determine a true or false value for the statement. So here, just as with any and-statement, if either M or N is false, the statement $M \,\&\, N$ is false. Otherwise, it's true.

Neither . . . nor

The *neither . . . nor* structure negates both parts of the statement. For example:

I like *neither* Mona *nor* Nunu.

This statement means *both* I don't like Mona *and* I don't like Nunu. Translate this statement into SL as

$$\sim M \,\&\, \sim N$$

Not . . . both

The *not . . . both* structure means that even though the statement as a whole is negated, either part by itself may not be negated. For example:

I do *not* like *both* Mona and Nunu.

This statement says that even though I don't like both people, I may like one of them. So you translate this statement into SL as

$$\sim(M \,\&\, N)$$

. . . if . . .

You already know how to translate a statement that starts with the word *if*. But this becomes more confusing when you find the word *if* placed in the middle of a statement, as in:

I like Mona *if* Chloe likes Olivia.

To clarify the statement, just untangle it as follows:

If Chloe likes Olivia, I like Mona.

With the statement reorganized, you can see that the way to translate it into SL is

$$O \rightarrow M$$

. . . only if . . .

This one (. . . only if . . .) is tricky until you think about it, and then it's really easy and you'll get it right every time. First, notice that the following statement is true:

Chloe lives in Louisville *only if* she lives in Kentucky.

This makes sense, because the only way that Chloe can live in Louisville is if she lives in Kentucky. Now, notice that the following statement is also true:

If Chloe lives in Louisville, then she lives in Kentucky.

This shows you that the two sentences are logically equivalent. So, when faced with an . . . *only if* . . . between two parts of a statement, just realize that it's an if–statement that's already in the correct order. Translate it into SL as

$L \rightarrow K$

. . . or . . .

As I mention in the section "Digging for or" earlier in the chapter, the little word *or* is big trouble. For example, it appears in this statement:

Chloe lives in Kentucky *or* Chloe likes Olivia.

Depending on how it's used, *or* can have two different meanings:

Chloe lives in Kentucky *or* Chloe likes Olivia, *or both.*

Chloe lives in Kentucky *or* Chloe likes Olivia, *but not both.*

Given *or*'s multiple personalities, my advice is this: When you see an *or* looking lonesome in a statement to be translated, it probably means that someone (like your professor) wants to make sure you know that, in logic, *or* always means *or . . . or both*. So translate the statement into SL as

$K \vee O$

. . . or . . . or both

The . . . *or . . . or both* structure is clear and easy: It says what it means and means what it says. For example:

Chloe lives in Kentucky *or* Chloe likes Olivia, *or both*.

Translate this statement in SL as

$K \vee O$

. . . or . . . but not both

The . . . *or . . . but not both* structure has a clear meaning, but it's not so easy to translate into SL. For example:

Chloe lives in Kentucky *or* Chloe likes Olivia, *but not both.*

In order to translate the words *but not both* into SL, you need to do some fancy logical footwork. As I discuss in the section "But, though, however, although, nevertheless . . ." earlier in the chapter, the word *but* becomes &, and the words *not both* are translated here as $\sim(K \,\&\, O)$. Putting it all together, you translate the whole statement into SL as

$$(K \vee O) \,\&\, \sim(K \,\&\, O)$$

Alternatively, you can use the if-and-only-if-operator, negated with parentheses:

$$\sim(K \leftrightarrow O)$$

IN THIS CHAPTER

- Evaluating SL statements
- Identifying a statement's main operator
- Knowing the eight forms of SL statements
- Recapping evaluation step-by-step

Chapter 5
Embracing the Value of Evaluation

People love simplicity.

Have you ever read halfway through a movie review and then skipped to the end to find out whether the movie got an easy-to-grasp thumbs-up or thumbs-down? Or have you paged through a car magazine to check out the simple 1-to-5-star rating that every car received? And surely you've never sat down with a friend to rate the appeal of people you both know on a scale of 1 to 10.

Movies, cars, and people are complicated. There's so much to understand about their makeup. But (it bears repeating) people love simplicity. I'm sure that you, like me, feel a sense of relief when you can reduce all that complexity down to something simple that you can easily make sense of. Logic evolved with this need in mind.

In Chapter 4, you discover how to take a complicated statement in English and write it with just a few symbols in sentential logic (SL). In this chapter, you take the process one step further by figuring out how to reduce a complicated statement in formal logic to a single truth value: either **T** or **F**. Like the thumbs-up or thumbs-down of movie reviews, life doesn't get much simpler than that.

This conversion process is called *evaluating a statement* or *computing a statement's truth value*. Whichever term you use, it's one of the key skills you need in your study of logic. After you understand this process, you'll be ahead of the curve on your next logic exam.

Evaluating Is the Bottom Line

An important aspect of SL is that it enables you to simplify complex statements through the process of *evaluation*. When you evaluate an SL statement, you replace all of its constants with truth values (**T** or **F**) and then reduce the statement to a *single truth value*. When you think *evaluation*, remember that the word means *to find the value of something*.

REMEMBER

Correctly evaluating statements is probably the most important skill to develop early on in your study of logic. Students who have trouble doing so can suffer for two reasons:

- **Evaluation is a key skill:** You'll probably get tested on evaluation multiple times — in homework assignments, quizzes, and exams. The study of logic becomes time-consuming and frustrating if you don't know how to evaluate statements well.
- **Evaluation is an important foundation for later skills:** More advanced applications of logic (such as truth tables) depend upon a solid understanding of evaluation, as you see in later chapters of this book.

Here's the good news: Evaluating is a plug-and-chug skill. It doesn't require cleverness or ingenuity. You just need to know the rules of the game, and then you need to practice, practice, practice.

Noticing how logic is similar to math

REMEMBER

The rules of the game for evaluating SL statements are a lot like the rules you probably already know for evaluating arithmetic problems. (Check out Chapter 4, where I outline other similarities between SL and arithmetic.) For example, take a look at this simple math problem:

$$5 + (2 \times (4 - 1)) = ?$$

To solve the problem, first evaluate what's inside the innermost set of parentheses. That is, because the value of $4 - 1$ is 3, you can replace $(4 - 1)$ with 3, and the problem becomes:

$$5 + (2 \times 3) = ?$$

Next, evaluate what's inside the remaining set of parentheses. This time, because the value of 2×3 is 6, you can make another replacement:

$$5 + 6 = ?$$

At this point, the problem is easy to solve. Because $5 + 6$ evaluates to 11, this value is the answer. Through a series of e*valu*ations, a string of numbers and symbols reduces to a single *value*.

Getting started with SL evaluation

When evaluating a statement in SL, the goal is the same as with an arithmetic problem: You want to evaluate the statement you're given, which means you need to find its value.

In the arithmetic problem I discuss in the preceding section, you already know the values of the four numbers (5, 2, 4, and 1). In an SL problem, you need to know the values of all the variables in the statement you wish to evaluate. That is, you need to know an *interpretation* for the statement.

REMEMBER

An *interpretation* of a statement in logic is a fixed set of truth values for all the constants in that statement.

For example, one possible interpretation for a statement is that P = **T**, Q = **F**, and R = **T**.

Remember that P, Q, and R are constants and that **T** and **F** are truth values. So, when I write P = **T**, this doesn't mean these two symbols are equal. Instead, it's a notation that means "the truth value of P is **T**."

You may not know for certain that this interpretation is correct, but you can still solve the problem under this interpretation — that is, under the assumption that the interpretation is correct. So, here's an SL evaluation problem:

> Under the interpretation P = **T**, Q = **F**, and R = **T**, evaluate the statement $\sim(\sim P \rightarrow (\sim Q \ \& \ R))$.

To solve this problem, begin by replacing each constant with its truth value as follows:

$$\sim(\sim \mathbf{T} \rightarrow (\sim \mathbf{F} \ \& \ \mathbf{T}))$$

TIP

After you replace the constants in an SL statement with their truth values, you technically don't have a statement anymore, and the purists may scowl. But while you're figuring out evaluation, turning SL statements into expressions of this kind is helpful.

The second and third not-operators are directly linked to truth values, which makes them easy to evaluate because the value of ~**T** is **F** and the value of ~**F** is **T**. (Check out Chapter 4 for a refresher on working with the logical operators you find in this chapter.) So, you can rewrite the expression as

$$\sim(\mathbf{F} \rightarrow (\mathbf{T} \& \mathbf{T}))$$

TIP

Parentheses in SL work just as they do in arithmetic: They break up an expression so that it's clear which part of the expression you need to figure out first. In this case, the innermost set of parentheses contains **T** & **T**. And because the value of **T** & **T** is **T**, this expression simplifies to

$$\sim(\mathbf{F} \rightarrow \mathbf{T})$$

Now, evaluate what's in the remaining set of parentheses. Because the value of **F** → **T** is **T**, the expression simplifies further to

$$\sim(\mathbf{T})$$

At this point, it's easy to see that ~**T** evaluates to **F**, which is the answer for the value of the expression. The result here is similar to the result of the arithmetic problem: You started with a complex statement and evaluated it by finding its value, which in logic is always either **T** or **F**.

Stacking up another method

The evaluation method I use in the previous section works for all statements in SL, no matter how complex they are. In the next example, I use the same method, but with a small cosmetic change: Instead of rewriting the entire equation at every step, I just accumulate truth values as I go along. Here's a new problem:

Evaluate $\sim(\sim P \ \& \ (\sim Q \leftrightarrow R))$ by using the interpretation P = **F**, Q = **T**, and R = **T**.

The first step is to replace the constants with their truth values. In the previous section's example, I replace all the constants and rewrite the whole equation. This time, I place the truth value for each constant directly below it.

$$\sim(\sim\underset{\mathbf{F}}{P} \;\&\; (\sim\underset{\mathbf{T}}{Q} \leftrightarrow \underset{\mathbf{T}}{R}))$$

This example has two not-operators that immediately precede constants. These operators are very easy to work with: Simply put the correct value under each operator, replacing the previous value.

$$\sim(\underset{\mathbf{T}}{\sim}P \;\&\; (\underset{\mathbf{F}}{\sim}Q \leftrightarrow \underset{\mathbf{T}}{R}))$$

WARNING

At this stage, don't try to evaluate any not-operator that immediately precedes an opening parenthesis. Because this operator negates *everything* inside the parentheses, you have to wait until you know the value of everything inside before you can put the operator into play.

Now you can work on the part of the expression that's inside the parentheses. Be sure to start on the innermost set. The operator you're evaluating here is the if-and-only-if-operator (↔). On one side of it, the value of ~*Q* is **F**. On the other side, the value of *R* is **T**. That gives you **F** ↔ **T**, which evaluates to **F**. Place this value directly under the operator you just evaluated, which is the if-and-only-if-operator. Doing so shows you that the value of everything inside the parentheses is **F**.

$$\sim(\underset{\mathbf{T}}{\sim}P \;\&\; (\sim Q \underset{\mathbf{F}}{\leftrightarrow} R))$$

Now move outward to the next set of parentheses. Here, you're evaluating the and-operator (&). On one side, the value of ~*P* is **T**. On the other side, the value of everything inside the inner parentheses is **F**. That gives you **T** & **F**, which evaluates to **F**. Place this value under the and-operator.

$$\sim(\sim P \underset{\mathbf{F}}{\&} (\sim Q \leftrightarrow R))$$

The final step is to evaluate the entire statement. The operator you're evaluating now is the not-operator that's outside the parentheses, and it negates everything inside the parentheses. In this case, it changes the value under the and-operator from **F** to **T**. Place this value under the first negation operator.

$$\underset{\mathbf{T}}{\sim}(\sim P \;\&\; (\sim Q \leftrightarrow R))$$

Now that you've evaluated everything in the expression, the last value, **T**, is the value of the whole statement. In other words, when *P* = **F**, *Q* = **T**, and *R* = **T**, the statement ~(~*P* & (~*Q* ↔ *R*)) evaluates to **T**. Notice how systematic evaluation enables you to turn a great deal of information into a single truth value.

Making a Statement

You get a taste of evaluation in the section "Evaluating *Is* the Bottom Line" earlier in the chapter, so now I give you a closer look at how SL statements work. After you understand a little bit more about these statements, you find that they practically evaluate themselves.

Identifying sub-statements

REMEMBER

A *sub-statement* is any piece of a statement that can stand on its own as a complete statement.

For example, the statement $P \vee (Q \,\&\, R)$ contains the following two sub-statements that can stand on their own as complete statements:

- $Q \,\&\, R$
- P

And here's an example of a piece of the statement $P \vee (Q \,\&\, R)$ that is *not* a sub-statement: $\vee (Q \,\&$. Even though it's a piece of the statement, it obviously isn't a complete statement in its own right. Instead, it's just a meaningless string of SL symbols. (In Chapter 13, you discover the fine points of how to tell a statement from a string of symbols.)

REMEMBER

When you evaluate a statement, you begin by placing values on the smallest possible sub-statements, which are the individual constants.

For example, suppose you want to evaluate the statement $P \vee (Q \,\&\, R)$ based on the interpretation P = **T**, Q = **T**, R = **F**. You begin by placing the truth value below each constant as follows:

$$\underset{\mathbf{T}}{P} \vee (\underset{\mathbf{T}}{Q} \,\&\, \underset{\mathbf{F}}{R})$$

Now you can evaluate the sub-statement $Q \,\&\, R$.

$$\underset{\mathbf{T}}{P} \vee (Q \,\underset{\mathbf{F}}{\&}\, R)$$

Finally, you can evaluate the whole statement.

$$P \underset{\mathbf{T}}{\vee} (Q \,\&\, R)$$

TIP

You evaluate a long statement by breaking it down — piece by piece — into sub-statements that you evaluate separately from smallest to largest.

Scoping out a statement

REMEMBER

After you know what a sub-statement is, it's easy to understand the *scope* of an operator, which refers to the smallest sub-statement that includes the operator in question.

For example, take the statement $(P \rightarrow (Q \vee R)) \leftrightarrow S$. Suppose you want to know the scope of the or-operator ($\vee$). Two possible sub-statements that include this operator are $P \rightarrow (Q \vee R)$ and $Q \vee R$. The shorter of the two is $Q \vee R$, so this sub-statement represents the scope of the operator.

TIP

You can also think of an operator's scope as the *range of influence* it holds over the statement.

To illustrate the range of influence, I've underlined the scope of the or-operator in the following statement:

$$(P \rightarrow (\underline{Q \vee R})) \leftrightarrow S$$

This shows that the or-operator affects, or influences, the constants Q and R, but *not* the constants P or S.

In contrast, I've underlined the scope of the if-operator ($\rightarrow$) in the same statement:

$$(\underline{P \rightarrow (Q \vee R)}) \leftrightarrow S$$

This shows that the if . . . then-operator's range of influence includes the constant P and the sub-statement $(Q \vee R)$, but not the constant S.

REMEMBER

Before you can evaluate an operator, you need to know the truth value of every other constant and operator in its scope. And after you understand how to find the scope of an operator, you can easily see why you need to begin evaluating a statement from inside the parentheses.

For example, in the statement $(P \rightarrow (Q \vee R)) \leftrightarrow S$, the or-operator is within the scope of the if-operator. This means that you can't evaluate the if-operator until you know the value of the or-operator.

WARNING

Be careful when figuring out the scope of the operators in statements with not-operators (~). The scope of a not-operator is always the smallest sub-statement that immediately follows it. When a not-operator is in front of a constant — think of it as being bound to the constant — its scope includes only that constant. For example, the scope of the first not-operator is underlined in the following statement:

$$\underline{\sim P} \ \& \sim (Q \& R)$$

In contrast, when a not-operator is in front of an opening parenthesis, its scope is everything inside that set of parentheses. For example, the scope of the second not-operator is underlined here:

$$\sim P \,\&\, \underline{\sim (Q \,\&\, R)}$$

The main attraction: Finding the main operator of a statement

REMEMBER

The *main operator* is the most important operator in a statement, for the following reasons:

- **Every SL statement has just one main operator.** This rule holds no matter how long or complicated a statement is.
- **The scope of the main operator is the whole statement.** Thus, the main operator affects every other constant and operator in the statement.
- **The main operator is the last operator that you evaluate.** This fact makes sense when you think about it: Because the scope of the main operator is *everything else* in the statement, you need to evaluate everything else before you can evaluate the main operator.
- **The main operator's value is the same as the value of the statement itself.** So, when you finally evaluate the main operator as either true or false, you know the value of the entire statement.
- **The main operator falls outside all parentheses, *except* when the whole statement includes an extra (and removable) set of parentheses.** I explain this in detail in the remainder of this section.

For example, suppose you want to evaluate $(P \rightarrow (Q \leftrightarrow R)) \,\&\, S$ under the interpretation P = **T**, Q = **F**, R = **F**, and S = **T**. Begin by assigning each constant its proper truth value.

$$(\underset{\mathbf{T}}{P} \rightarrow (\underset{\mathbf{F}}{Q} \leftrightarrow \underset{\mathbf{F}}{R})) \,\&\, \underset{\mathbf{T}}{S}$$

Next, evaluate the sub-statement $Q \leftrightarrow R$ to get the value of the if-and-only-if–operator.

$$(\underset{\mathbf{T}}{P} \rightarrow (Q \underset{\mathbf{T}}{\leftrightarrow} R)) \,\&\, \underset{\mathbf{T}}{S}$$

Doing this allows you to evaluate $P \rightarrow (Q \leftrightarrow R)$ to get the value of the if-operator.

$$(P \underset{\mathbf{T}}{\rightarrow} (Q \leftrightarrow R)) \,\&\, \underset{\mathbf{T}}{S}$$

And finally, you can evaluate the whole statement, which gives you the value of the statement's main operator, the and-operator.

$$(P \rightarrow (Q \leftrightarrow R)) \underset{\mathbf{T}}{\&} S$$

Therefore, the truth value of this statement is **T**, which is also the truth value of its main operator.

Because the main operator is so important, you need to be able to pick it out of any statement. Doing so is usually quite simple when you apply a few quick rules. Every SL statement falls into one of three cases, which I outline in the following sections.

REMEMBER

If you come across a statement that doesn't fall into one of these cases, it isn't well-formed, which means it really isn't a statement at all. I discuss this idea further in Chapter 14. For now, though, any statement you run across will have a main operator that you can find without difficulty.

When only one operator is outside the parentheses

Sometimes, it's easy to find the main operator because it's the *only* operator outside all the parentheses. For example, take a look at this statement:

$$(P \vee \sim Q) \,\&\, (R \rightarrow P)$$

The main operator here is the and-operator. Similarly, check out this statement:

$$\sim (P \,\&\, (Q \leftrightarrow R))$$

The main operator here is the not-operator.

When no operator is outside the parentheses

TIP

If you find *no* operator outside the parentheses in a statement, then the outer set of parentheses is unnecessary. Take this statement, for example:

$$((\sim P \leftrightarrow Q) \rightarrow R)$$

Notice that the outer set of parentheses really isn't necessary. Removing the set gives you this statement:

$$(\sim P \leftrightarrow Q) \rightarrow R$$

Now, the only operator outside the parentheses is the if-operator, which is indeed the main operator.

Note: In this book, I avoid using unnecessary parentheses because they take up space and don't add anything useful to a statement. (In Chapter 14, I discuss the nitty-gritty details of why a statement may contain extra parentheses.)

When more than one operator is outside the parentheses

In some statements, you may find more than one operator outside the parentheses. For example

$$\sim(\sim P \rightarrow Q) \vee \sim(P \rightarrow Q)$$

REMEMBER

When more than one operator is outside the parentheses, the main operator is always the one that *isn't* a not-operator.

In the preceding example, the main operator is the or-operator.

Recognizing the Eight Forms of SL Statements

In SL, a variable can stand for an entire statement (or sub-statement). You can use variables to classify SL statements into eight different *statement forms*, which are generalized versions of SL statements. Table 5-1 shows the eight basic statement forms.

TABLE 5-1 The Eight Forms of SL Statements

Positive Forms	Negative Forms
$x \,\&\, y$	$\sim(x \,\&\, y)$
$x \vee y$	$\sim(x \vee y)$
$x \rightarrow y$	$\sim(x \rightarrow y)$
$x \leftrightarrow y$	$\sim(x \leftrightarrow y)$

To illustrate how these statement forms work, here are three statements whose main operators are all and-operators:

$$P \,\underline{\&}\, Q \qquad (P \vee \sim Q) \,\underline{\&}\, \sim(R \rightarrow S) \qquad (((\sim P \leftrightarrow Q) \rightarrow R) \vee (\sim Q \,\&\, S)) \,\underline{\&}\, R$$

PARTS OF THE WHOLE

Personally, I find the following terminology for the various parts of a statement to be a bit over the top. If your professor wants you to know these terms, you're going to have to memorize them. But I think the important thing — when you have an SL statement to evaluate — is that you can find the main operator and pick out which of the eight forms the statement takes. When speaking about the various parts of a statement, it's just as easy to use the words *first part* and *second part*.

Here are some quick rules to help you understand the statement jargon:

- When a statement is in the form $x \,\&\, y$ it's an and-statement, which is also called a *conjunction*. In this case, both parts of the statement are called *conjuncts*.
- When a statement is in the form $x \vee y$, it's an or-statement, which is also called a *disjunction*. In this case, both parts of the statement are called *disjuncts.*
- When a statement is in the form $x \rightarrow y$, it's an if-statement, which is also called an *implication*. In this case, the first part of the statement is called the *antecedent*, and the second part is called the *consequent*.
- When a statement is in the form $x \leftrightarrow y$, it's an if-and-only-if–statement, which is also called a *double implication*. In this case, the two parts of the statement actually don't have special names. (Perhaps you can convince your logic professor to give extra credit to the student who makes up the best names?)

Although all three of these statements are obviously different, you can represent each of them by using the following statement form:

$$x \,\&\, y$$

For example, in the statement $P \,\&\, Q$, the variable x stands for the sub-statement P, and the variable y stands for the sub-statement Q. Similarly, in the statement $(P \vee \sim Q) \,\&\, \sim(R \rightarrow S)$, x stands for the sub-statement $(P \vee \sim Q)$, and y stands for the sub-statement $\sim(R \rightarrow S)$. And finally, in the statement $(((\sim P \leftrightarrow Q) \rightarrow R) \vee (\sim Q \,\&\, S)) \,\&\, R$, x stands for $(((\sim P \leftrightarrow Q) \rightarrow R) \vee (\sim Q \,\&\, S))$ and y stands for R.

When a statement's main operator is one of the four binary operators (&, $\vee$, $\rightarrow$, or $\leftrightarrow$), its statement form is one of the four positive forms in Table 5-1. However, when a statement's main operator is the not-operator, its form is one of the negative forms in Table 5-1. To find out which one, you need to look at the operator with the next-widest scope. Here's an example:

$$\sim((P \rightarrow Q) \leftrightarrow (Q \vee R))$$

In this case, the main operator is the not-operator. The if-and-only-if–operator ($\leftrightarrow$) has the next widest scope, covering everything inside the outermost parentheses. So, you can represent this statement using this statement form:

$$\sim(x \leftrightarrow y)$$

This time, the variable *x* stands for the sub-statement $(P \rightarrow Q)$, and the variable *y* stands for the sub-statement $(Q \vee R)$.

TIP

Recognizing the basic form of a given statement is a skill you use in other chapters. For now, be aware that you can represent every statement by one of the eight basic statement forms.

Revisiting Evaluation

I hope the preceding chapter sections help you make sense of statement evaluation. I think you're less likely to make mistakes while evaluating an SL statement when you understand how all the pieces of that statement fit together.

For example, suppose that you want to evaluate $\sim(\sim(P \vee Q) \ \& \ (\sim R \leftrightarrow S))$ under the interpretation P = **T**, Q = **F**, R = **F**, and S = **T**. The statement looks complicated, but you can apply the ideas from earlier in the chapter to be up for the challenge!

Before you begin, look over the statement. It's in the form $\sim(x \ \& \ y)$, with the first part of the statement being $\sim(P \vee Q)$ and the second part being $(\sim R \leftrightarrow S)$. You need to get the truth value of both parts before you can evaluate the and-operator that comes between them. Only then can you evaluate the first not-operator, which is the statement's main operator.

Follow these steps to determine the statement value for the given interpretation (P = **T**, Q = **F**, R = **F**, and S = **T**):

1. **Place the appropriate truth values under the constants in the statement.**

   You now have:

   $$\sim(\sim(\underset{\mathbf{T}}{P} \vee \underset{\mathbf{F}}{Q}) \ \& \ (\sim\underset{\mathbf{F}}{R} \leftrightarrow \underset{\mathbf{T}}{S}))$$

2. **Write the value of the not-operator in front of the constant *R*.**

   You now have:

   $$\sim(\sim(\underset{\mathbf{T}}{P} \vee \underset{\mathbf{F}}{Q}) \ \& \ (\underset{\mathbf{T}}{\sim}R \leftrightarrow \underset{\mathbf{T}}{S}))$$

3. **Determine the values of both the or-operator and the if-and-only-if–operator.**

   You now have:

   $$\sim(\sim(P \underset{\mathbf{T}}{\vee} Q) \,\&\, (\sim R \underset{\mathbf{T}}{\leftrightarrow} S))$$

4. **Find the value of the sub-statement that takes the $\sim(x \vee y)$ form.**

   You may be tempted at this point to evaluate the and-operator, but first you need the value of the sub-statement $\sim(P \vee Q)$, which means you need to get the value of the not-operator.

   $$\sim(\underset{\mathbf{F}}{\sim}(P \vee Q) \,\&\, (\sim R \underset{\mathbf{T}}{\leftrightarrow} S))$$

5. **Evaluate the and-operator.**

   You get:

   $$\sim(\sim(P \vee Q) \underset{\mathbf{F}}{\&} (\sim R \leftrightarrow S))$$

6. **Evaluate the main operator (the first not- operator).**

   After you've evaluated every other operator in the statement, you can evaluate the first not-operator.

   You get:

   $$\underset{\mathbf{T}}{\sim}(\sim(P \vee Q) \,\&\, (\sim R \leftrightarrow S))$$

The truth value of the main operator is the value of the whole statement, so you know that under the given interpretation, the statement is true.

IN THIS CHAPTER

- Creating and analyzing statements by using truth tables
- Applying truth tables to evaluate tautologies, contradictions, and contingencies
- Seeing the connections between statement types and evaluations

Chapter 6

Turning the Tables: Evaluating Statements with Truth Tables

In this chapter, you discover one of the most important tools in sentential logic (SL): the *truth table*. Truth tables enable you to evaluate a statement under every possible *interpretation* (a fixed set of truth values for all statement constants; see Chapter 5), which in turn provides insight about a statement even when you don't know the truth values of its constants.

Truth tables open up vast new logical territory. First of all, they're an easy way to find out when a statement is true or false — a central question in logic. But beyond this, they can help you identify *tautologies* and *contradictions* (statements in SL that are always true or always false).

You can also use truth tables to decide whether a set of statements is *consistent* — that is, whether all the interpretations can be true. Additionally, with truth tables, you can figure out whether two statements are *semantically equivalent* — that is, whether they have the same truth value in all possible cases. Finally, and perhaps most importantly, truth tables provide a way to determine the *validity* of an argument — that is, whether that argument is valid or invalid.

Applying truth tables in to SL problems is where the rubber meets the road, so fasten your seat belt!

Putting It All on the Table: The Joy of Brute Force

Sometimes, solving a problem requires cleverness and ingenuity. Before you can get the answer, you need to have that "Aha!" moment that allows you to see things in a whole new way. "Aha!" moments can be exhilarating, but also frustrating, especially if you're working on an exam under time pressure and the "Aha!" is nowhere in sight.

Truth tables are the antidote to "Aha!" They rely on a method that mathematicians informally call *brute force*. In this type of approach, instead of trying to find that one golden path to success, you doggedly exhaust all possible paths. Brute force methods can be time-consuming, but at the end of the day, you can always find the answer you're looking for.

For example, suppose on your first exam you encounter this cryptic question:

> What can you say about this statement: $P \rightarrow (\sim Q \rightarrow (P \ \& \sim Q))$? Justify your answer.

With the clock ticking, you can probably think of many things you want to say about the statement, but they're not going to get you any points on the test. So, you stare at it, waiting for the "Aha!" to arrive. And suddenly it does. You finally think about it this way:

> This statement is just stating the obvious: "If I assume *P* is true and then assume that *Q* is false, I can conclude that *P* is true and *Q* is false." So, the statement is *always* true.

That wasn't so bad. Still, what if the "Aha!" never arrived? What if you weren't sure how to "justify your answer"? Or, worst of all, what if the question you faced looked like this:

> What can you say about this statement:
>
> $((\sim P \vee Q) \rightarrow ((R \ \& \sim S) \vee P)) \rightarrow (\sim Q \vee ((\sim R \vee S) \rightarrow P)) \rightarrow ((P \ \& \sim Q) \vee (S \rightarrow P))$?
>
> Justify your answer.

Facing complex statements like these signal it's time to use brute force, and truth tables are the ticket.

Constructing Baby's First Truth Table

Using a truth table gives you a way of organizing information in SL. It enables you to set up every possible combination of values for every constant (every interpretation) and see what happens when you evaluate the statement in each case. Chapter 4 presents some small truth tables that I use to introduce each of the five SL operators. These truth tables show how to interpret all possible input values to obtain an output value.

In the following sections, I show you how to set up, fill in, and draw conclusions from a truth table by working through the statement $P \rightarrow (\sim Q \rightarrow (P \,\&\, \sim Q))$.

Setting up a truth table

REMEMBER

A truth table is a way to organize *every possible interpretation* of a statement into horizontal rows, allowing you to evaluate the statement under all these interpretations.

Setting up a truth table is as simple as this four-step process:

1. **Set up the top row of the truth table with each constant on the left and the statement on the right.**

   Here's how you set up the first row for the statement you have to work with:

P	Q	P	$\rightarrow$	$(\sim$	Q	$\rightarrow$	$(P$	$\&$	$\sim$	$Q))$

2. **Determine the number of additional rows your table needs, based on the number of constants in the statement.**

TIP

A truth table needs one row for every possible interpretation for the given statement. To figure out how many rows you need, multiply the number 2 by itself one time for every constant in the statement. Because the statement $P \rightarrow (\sim Q \rightarrow (P \,\&\, \sim Q))$ has two constants, you'll need four rows.

To make sure you're on the right track, check out Table 6-1, which contains the number of rows you need for statements that have one to five constants.

TABLE 6-1 Number of Constants and Rows in a Truth Table

Number of Constants	Constants	Number of Interpretations (Rows in the Truth Table)
1	*P*	2
2	*P* and *Q*	2 x 2 = 4
3	*P*, *Q*, and *R*	2 x 2 x 2 = 8
4	*P*, *Q*, *R*, and *S*	2 x 2 x 2 x 2 = 16
5	*P*, *Q*, *R*, *S*, and *T*	2 x 2 x 2 x 2 x 2 = 32

3. Set up the constant columns so that every possible combination of truth values is accounted for.

TIP

A good way to fill in these columns with **T**s and **F**s is to start in the right-hand constant column (in this case, the *Q* column) and fill that column in by alternating **T**s and **F**s — **T F T F** — all the way down. Then move one column to the left and fill that column in, alternating the **T**s and **F**s by twos: **T T F F**. If there are more columns (for example, because you have three or four constants in a statement), continue moving to the left and alternating by fours (**T T T T F F F F** . . .), eights (**T T T T T T T T F F F F F F F F** . . .), and so on.

So, in this example, alternate **T F T F** under the *Q* column, and then **T T F F** under the *P* column:

P	*Q*	*P*	→	(~	*Q*	→	(*P*	&	~	*Q*))
T	**T**									
T	**F**									
F	**T**									
F	**F**									

4. Draw horizontal lines between rows, and draw vertical lines separating all constants and operators in the statement.

P	*Q*	*P*	→	(~	*Q*	→	(*P*	&	~	*Q*))
T	**T**									
T	**F**									
F	**T**									
F	**F**									

I suggest this step for three reasons: First, the table starts out neat so you don't get confused. Second, you'll know the table is finished when all the little boxes are filled in with **T**s and **F**s. And third, the completed table is clear and readable. (If you draw the lines nicely with a ruler, you'll melt the heart of even the iciest professor.)

REMEMBER

When creating a truth table, you don't need separate columns for the parentheses, but be sure to bunch each parenthesis in with the constant or operator that immediately *follows* it, except for those at the end of a statement or sub-statement.

Filling in a truth table

Every row of your truth table accounts for a different interpretation of the statement. Filling in the truth table now simply becomes the process of evaluating the statement under every interpretation (in this case, under all four interpretations).

In Chapter 4, I discuss how to evaluate a statement from the inside out. The rules are still the same when you construct a truth table, but now you need to work every step of the process on every row of the table.

TIP

The step-by-step procedure I show you here fills in the table by working down the columns because it's easier and faster than trying to evaluate row by row.

As I work through the steps in this example, note that in each of the following truth tables:

- The columns containing the new truth values that I fill in during the step are in boldface.
- The columns containing the input values for that step (the values I'm using to get the new truth values) are in underlined type.

1. **Copy the value of each constant from the first column into the proper statement columns for those constants.**

P	*Q*	*P*	→	(~	*Q*	→	(*P*	&	~	*Q*))
T	T	**T**			**T**		**T**			**T**
T	F	**T**			**F**		**T**			**F**
F	T	**F**			**T**		**F**			**T**
F	F	**F**			**F**		**F**			**F**

Just copying. Pretty simple, right?

2. **In each column that has a not-operator (~) directly in front of a constant, write the corresponding negation of that constant in each row.**

P	*Q*	*P*	→	(~	*Q*	→	(*P*	&	~	*Q*))
T	T	T		**F**	T		T		**F**	T
T	F	T		**T**	F		T		**T**	F
F	T	F		**F**	T		F		**F**	T
F	F	F		**T**	F		F		**T**	F

WARNING

Make sure that every not-operator is in front of a constant. If it's in front of an opening parenthesis, it negates the entire value of everything inside the parentheses. In that case, you have to wait until you know the value of what's in the parentheses.

As you can see, this step isn't much more difficult than the previous one.

3. **Starting with the innermost set of parentheses, fill in the column directly below the operator for that part of the statement.**

Step 3 is really the meat and potatoes of truth tables. The good news is that with practice you're going to get really fast at this stage of filling in tables.

The innermost parentheses in this example contain the statement *P* & ~*Q*. The operator you're using to evaluate is the and-operator (&), and the two input values are in the columns under *P* and the not-operator.

1. In the first row, the value under *P* is **T** and the value under the not-operator is **F**. And **T** & **F** evaluates to **F**, so write this value in this row directly under the and-operator.
2. Repeat Sub-Step 1 for the other three rows, and your table should look like this:

P	*Q*	*P*	→	(~	*Q*	→	(*P*	&	~	*Q*))
T	T	T		F	T		T	**F**	F	T
T	F	T		T	F		T	**T**	T	F
F	T	F		F	T		F	**F**	F	T
F	F	F		T	F		F	**F**	T	F

4. Repeating Step 3, work your way outward from the first set of parentheses until you've evaluated the main operator for the statement.

1. The next operator you're using to evaluate is the if-operator (→) inside the outermost parentheses. The two input values are in the columns under the first not-operator and the and-operator.

   For example, in the first row, the value under the not-operator is **F** and the value under the and-operator is **F**. And **F** → **F** evaluates to **T**, so write this value in this row directly under the if-operator.

2. Repeat Sub-Step 1 to complete the column for the if-operator, and your table should look like this:

P	*Q*	*P*	→	(~	*Q*	→	(*P*	&	~	*Q*))
T	T	T		F	T	**T**	T	F	F	T
T	F	T		T	F	**T**	T	T	T	F
F	T	F		F	T	**T**	F	F	F	T
F	F	F		T	F	**F**	F	F	T	F

   Now you're ready to move to the main operator, the if-operator that's outside the parentheses (flip back to Chapter 5 for information on how to determine the main operator).

   The two input values are in the column under the *P* and the column under the other if-operator.

3. Fill in the first row for the main operator by using the value under the *P* (which is **T**) and the value under the if-operator inside the parentheses (which is **T**). And **T** → **T** evaluates to **T**, so write this result directly under the main if-operator in this row.

   Repeating Sub-Step 3 to finish up the column gives you:

P	*Q*	*P*	→	(~	*Q*	→	(*P*	&	~	*Q*))
T	T	T	**T**	F	T	T	T	F	F	T
T	F	T	**T**	T	F	T	T	T	T	F
F	T	F	**T**	F	T	T	F	F	F	T
F	F	F	**T**	T	F	F	F	F	T	F

WARNING

When you're creating a truth table, the column under the main operator should be the last column you fill in. If it isn't, you'd better erase (you were using pencil, weren't you?) and retrace your steps!

Reading a truth table

TIP

Circle the entire column under the main operator, so the information jumps out at you when it's time to read the table. The column under the main operator is the most important column in the table. It tells you the truth value for each interpretation of the statement.

In the example from the previous section, if you want to know how the statement evaluates when P is false and Q is true, just check the third row of the table. The value in this row under the main operator is **T**, so when P is false and Q is true, the statement evaluates as true.

At this point, you can go back to the original question with great confidence:

> What can you say about this statement: $P \rightarrow (\sim Q \rightarrow (P \,\&\, \sim Q))$? Justify your answer.

With your trusty truth table, you can tell your professor exactly what they want to hear: that the statement is always true, regardless of the value of the constants P and Q.

And how do you justify your answer? You don't have to because the truth table does it for you. As long as you filled it in correctly, the table tracks every possible interpretation of the statement. No other interpretations are possible, so your work is complete.

Putting Truth Tables to Work

After you know how to use truth tables, you can begin to understand SL at a whole new level. In this section, I show you how to tackle a few common problems involving individual statements, pairs and sets of statements, and arguments. (In later chapters, I show you how to tackle these same questions using a variety of other tools.)

Taking on tautologies and contradictions

REMEMBER

Every statement in SL falls into one of these three categories: *tautologies* (true under every interpretation), *contradictions* (false under every interpretation), or *contingent statements* (either true or false, depending upon the interpretation).

In the section "Constructing Baby's First Truth Table" earlier in the chapter, you see how you can use truth tables to sort out the truth value of a statement under every possible interpretation, which allows you to divide statements into three categories:

- **Tautologies:** Statements that are always true, regardless of the truth values of their constants. An example of a tautology is the statement $P \vee \sim P$ Because either P or $\sim P$ is true, at least one part of this or-statement is true, so the statement is always true.
- **Contradictions:** Statements that are always false, regardless of the truth values of their constants. An example of a contradiction is the statement $P \,\&\, \sim P$. Because either P or $\sim P$ is false, at least one part of this and-statement is false, so the statement is always false.
- **Contingent statements:** Statements that are true under at least one interpretation and false under at least one interpretation. An example of a contingent statement is $P \rightarrow Q$. This statement is true when P is true and Q is true, but false when P is true and Q is false.

Truth tables are an ideal tool for deciding which category a particular statement falls into. For example, in the section "Constructing Baby's First Truth Table" earlier in this chapter, you use a truth table to show that the statement $P \rightarrow (\sim Q \rightarrow (P \,\&\, \sim Q))$ evaluates as true in every row of the truth table, so the statement is a tautology.

WARNING

Don't make the mistake of thinking that every statement is either a tautology or a contradiction. You can find plenty of contingent statements that don't fall into either category.

Judging semantic equivalence

You can use truth tables to evaluate single statements under all possible combinations of truth values for their constants. But you can also take this process a step further and compare two statements at a time.

REMEMBER

When two statements are *semantically equivalent,* they have the same truth value under all interpretations.

You already know a simple example of two statements that are semantically equivalent: P and $\sim\sim P$. When the value P is **T**, the value of $\sim\sim P$ is also **T**. Similarly, when the value of P is **F**, the value of $\sim\sim P$ is also **F**.

This example is easy to check because, with only one constant, this statement has only two possible interpretations. As you can probably imagine, the more constants you have, the more complex the task of determining semantic equivalence becomes.

However, a truth table can help you break down the complexity. For example, are the statements $P \rightarrow Q$ and $\sim P \vee Q$ semantically equivalent? You can find out by making a truth table for these two statements.

P	Q	P	$\rightarrow$	Q	$\sim$	P	$\vee$	Q
T	**T**							
T	**F**							
F	**T**							
F	**F**							

As I state in the section "Constructing Baby's First Truth Table" earlier in the chapter, the first step for filling out the truth table is to copy the value of each constant into the proper columns:

P	Q	P	$\rightarrow$	Q	$\sim$	P	$\vee$	Q
T	T	**T**		**T**		**T**		**T**
T	F	**T**		**F**		**T**		**F**
F	T	**F**		**T**		**F**		**T**
F	F	**F**		**F**		**F**		**F**

Second, handle any not-operator that applies directly to a constant:

P	*Q*	*P*	→	*Q*	~	*P*	∨	*Q*
T	T	T		T	**F**	T		T
T	F	T		F	**F**	T		F
F	T	F		T	**T**	F		T
F	F	F		F	**T**	F		F

Third, complete the evaluations of both statements separately, just as you would for a single statement, by filling in the value for the main operator:

P	*Q*	*P*	→	*Q*	~	*P*	∨	*Q*
T	T	T	**T**	T	F	T	**T**	T
T	F	T	**F**	F	F	T	**F**	F
F	T	F	**T**	T	T	F	**T**	T
F	F	F	**T**	F	T	F	**T**	F

REMEMBER

When two statements have the same truth value for the main operator on every row of the truth table, they're semantically equivalent. Otherwise, they're not.

The table shows that, in this case, the two statements are semantically equivalent. (Chapter 8 explains how you can apply this important concept of semantic equivalence to proofs in SL.)

Staying consistent across statements

If you can compare two statements at a time, why not more than two? In this section, you see how to construct a truth table that contains more than two statements.

REMEMBER

When a set of statements is *consistent*, at least one interpretation makes all of those statements true. When a set of statements is *inconsistent*, no interpretation makes all of them true.

For example, look at the following set of statements:

$P \vee \sim Q$

$P \rightarrow Q$

$P \leftrightarrow \sim Q$

Now, consider whether all three of these statements can be true at the same time. To find out, you must discover at least one combination of truth values for the constants *P* and *Q* that causes all three statements to be evaluated as true.

Again, a truth table is the tool of choice. This time, however, you put all three statements on the truth table. Here, I show you the table with the correct information filled in for the first statement. Specifically, I

- Copy the truth values of *P* and *Q* for each row into the correct columns.
- Evaluate the truth value of ~*Q*.
- Evaluate the whole statement $P \vee \sim Q$, placing the value for each row under the main operator (∨).

In three of the four rows, the first statement evaluates as true. But when *P* is false and *Q* is true, the statement is false. Because you're looking for at least one row in which all three statements are true, you can rule out the crossed-out row.

P	*Q*	*P*	∨	~	*Q*	*P*	→	*Q*	*P*	↔	~	*Q*
T	T	T	**T**	F	T							
T	F	T	**T**	T	F							
F	T	F	**F**	F	T							
F	F	F	**T**	T	F							

TIP

When filling in a truth table to test for consistency, move vertically like you do for a single statement, but evaluate one statement at a time. When you find a row where the statement evaluates as false, draw a line through to the rightmost column of the table. Drawing a line through the row saves you a few steps by reminding you not to evaluate any other statement in that row.

Repeating this process for the other two statements gives you the following result:

P	*Q*	*P*	∨	~	*Q*	*P*	→	*Q*	*P*	↔	~	*Q*
T	T	T	**T**	F	T	T	**T**	T	T	**F**	F	T
T	F	T	**T**	T	F	T	**F**	F				
F	T	F	**F**	F	T							
F	F	F	**T**	T	F	F	**T**	F	F	**F**	T	F

REMEMBER

When every row of the truth table has at least one statement that evaluates as false, the statements are inconsistent. Otherwise, they're consistent.

In this case, you know that the three statements are inconsistent because under all interpretations, at least one statement is false.

Arguing with validity

REMEMBER

As I discuss in Chapter 3, for a valid argument, when all the premises are true, the conclusion must also be true. You can define the same basic idea in terms of interpretations.

- When an argument is *valid*, no interpretation exists for which all the premises are true and the conclusion is false.
- When an argument is *invalid*, however, at least one interpretation exists for which all the premises are true and the conclusion is false.

You can also use truth tables to decide whether an entire argument is valid. For example, consider this argument in which you have

These premises:

$$P \,\&\, Q$$

$$R \rightarrow {\sim} P$$

And this conclusion:

$${\sim} Q \leftrightarrow R$$

In this case, the argument has three constants — *P*, *Q*, and *R* — so the truth table needs to have eight rows, because $2 \times 2 \times 2 = 8$ (refer to Table 6-1). You can set up the truth table for these argument statements like the truth table in the preceding section, with the constants on the left followed by the three statements across the top row.

TIP

To set up a large truth table, start in the rightmost constant column, which appears on the left side of the table. (The rightmost constant is *R* in this table.) Moving down the column, write **T**, **F**, **T**, **F**, and so on, alternating values every other row until you finish the last row. Then move one constant column to the left and write **T**, **T**, **F**, **F**, and so on, alternating values every two rows. Keep moving left in the constant columns and doubling the occurrences of values — alternating the next constant column by fours, then by eights, and so forth until the table is complete.

Here's the truth table for this argument with the constant columns set up:

P	*Q*	*R*	*P*	&	*Q*	*R*	→	~	*P*	~	*Q*	↔	*R*
T	**T**	**T**											
T	**T**	**F**											
T	**F**	**T**											
T	**F**	**F**											
F	**T**	**T**											
F	**T**	**F**											
F	**F**	**T**											
F	**F**	**F**											

In the preceding section, I mention the advantage of evaluating all rows for an entire statement before moving on to the next statement. Here's how the table looks after you evaluate the first statement:

P	Q	R	P	&	Q	R	→	~	P	~	Q	↔	R
T	T	T	T	**T**	T								
T	T	F	T	**T**	T								
T	F	T	T	**F**	F								
T	F	F	T	**F**	F								
F	T	T	F	**F**	T								
F	T	F	F	**F**	T								
F	F	T	F	**F**	F								
F	F	F	F	**F**	F								

TIP

When you find a row where either a premise evaluates as false or the conclusion evaluates as true, draw a line all the way through it. Drawing a line through the row saves you a few steps by reminding you not to evaluate any other statement in this row.

The first evaluation of this example is especially helpful because in six of the eight rows of the truth table, the first premise is false, which means you can rule out these six rows. Here's what the rest of the table looks like when completed:

P	Q	R	P	&	Q	R	→	~	P	~	Q	↔	R
T	T	T	T	**T**	T	T	**F**	F	T				
T	T	F	T	**T**	T	F	**T**	F	T	F	T	**T**	F
T	F	T	T	**F**	F								
T	F	F	T	**F**	F								
F	T	T	F	**F**	T								
F	T	F	F	**F**	T								
F	F	T	F	**F**	F								
F	F	F	F	**F**	F								

REMEMBER

When no row of the truth table contains all true premises and a false conclusion, the argument is valid; otherwise, it's invalid.

As you can see, the only row in the previous table that has all true premises also has a true conclusion, so this argument is valid.

Building New Tautologies and Contradictions

The previous sections of this chapter show you how to use truth tables to test for a variety of logical conditions. Table 6-2 organizes this information.

TABLE 6-2 Truth Table Tests for a Variety of Logical Conditions

Condition Being Tested	Number of Statements	Condition Verified When
Tautology	1	The statement is true in every row.
Contradiction	1	The statement is false in every row.
Contingent Statement	1	The statement is true in at least one row and false in at least one row.
Semantic Equivalence	2	Both statements have the same truth value in every row.
Semantic Inequivalence	2	The two statements have different values in at least one row.
Consistency	2 or more	All statements are true in at least one row.
Inconsistency	2 or more	All statements aren't true in any row.
Validity	2 or more	In every row where all premises are true, the conclusion is also true.
Invalidity	2 or more	All premises are true *and* the conclusion is false in at least one row.

If you have a sneaking suspicion that all these concepts are somehow connected, you're right. This section shows you how many of the concepts in this chapter can be recast as either tautologies or contradictions.

Turning a tautology into a contradiction, and vice versa

REMEMBER

You can easily turn a tautology into a contradiction (or vice versa) by negating the whole statement with a not-operator (~).

Recall from the section "Taking on tautologies and contradictions" earlier in the chapter, that the statement

$$P \rightarrow (\sim Q \rightarrow (P \& \sim Q))$$

is a tautology. So its negation, the statement

$$\sim(P \rightarrow (\sim Q \rightarrow (P \& \sim Q)))$$

is a contradiction.

Similarly, you can turn a contradiction into a tautology in the same way (by negating the whole contradiction by adding the not-operator). Thus, the statement

$$\sim(\sim(P \rightarrow (\sim Q \rightarrow (P \& \sim Q))))$$

is a tautology. This connection shows you that even though tautologies and contradictions are polar opposites, they're very closely linked.

Changing a pair of semantically equivalent statements into a tautology

REMEMBER

When you connect any two semantically equivalent statements with the if-and-only-if–operator ($\leftrightarrow$), the resulting statement is a tautology.

As you find in the section "Judging semantic equivalence" earlier in the chapter, the two statements

$$P \rightarrow Q \qquad\qquad \sim P \vee Q$$

are semantically equivalent. That is, no matter which truth values you choose for P and Q, the two statements have the same evaluation.

Now connect these two statements with an if-and-only-if–operator:

$$(P \rightarrow Q) \leftrightarrow (\sim P \vee Q)$$

The result is a tautology. And, as you may deduce from the previous section, you can turn this tautology into a contradiction by negating it as follows:

$$\sim((P \rightarrow Q) \leftrightarrow (\sim P \vee Q))$$

Converting a set of inconsistent statements into a contradiction

REMEMBER

When you connect an inconsistent set of statements in a single statement by repeated use of the and-operator (&), the resulting statement is a contradiction.

As you find in the section "Staying consistent across statements" earlier in the chapter, the three statements

$$P \vee {\sim}Q \qquad\qquad P \rightarrow Q \qquad\qquad P \leftrightarrow {\sim}Q$$

are inconsistent. That is, under any interpretation, at least one of the statements is false.

Now connect these three statements with and-operators:

$$((P \vee {\sim}Q)\ \&\ (P \rightarrow Q))\ \&\ (P \leftrightarrow {\sim}Q)$$

REMEMBER

When you use operators to connect more than two statements, you need to use extra parentheses so it's clear which operator is the main operator. I discuss this in more detail in Chapter 14.

In this connected statement, the main operator is the second and-operator — the only operator outside the parentheses — but in any case, the result is a contradiction.

Recasting a valid argument as a contradiction

As you may have guessed, argument validity can also be woven into this tapestry: To change a valid argument to an inconsistent statement, connect the premises and the *negation* of the conclusion repeatedly with and-operators. For example, here's a simple example of a valid argument:

Premises:

$$P \rightarrow Q$$

$$Q \rightarrow R$$

Conclusion:

$$P \rightarrow R$$

Because this argument's valid, you know it's impossible that both premises are true and the conclusion is false. So, if you negate the conclusion by using a not-operator (~), the result is a set of inconsistent statements:

$$P \to Q \qquad Q \to R \qquad \sim(P \to R)$$

Thus, as you discover in the previous section, you can turn this set of statements into a contradiction by connecting them with and-operators:

$$((P \to Q) \,\&\, (Q \to R)) \,\&\, \sim(P \to R)$$

Table 6-3 encapsulates all of the connections from this section into a single resource for reference:

TABLE 6-3 **Building New Tautologies and Contradictions**

To Change. . .	To. . .	Do This. . .
A tautology	A contradiction	Negate it by attaching a not-operator.
A contradiction	A tautology	Negate it by attaching a not-operator.
Two semantically equivalent statements	A tautology	Connect the statements with an if-and-only-if operator.
A set of inconsistent statements	A contradiction	Connect the statements with and-operators.
A valid argument	A contradiction	Connect the premises and the ***negation*** of the conclusion with and-operators

IN THIS CHAPTER

- Looking at the truth values of whole statements
- Understanding how to set up, fill in, and read a quick table
- Knowing which type of statements you're working with

Chapter 7

Taking the Easy Way Out: Creating Quick Tables

In Chapter 6, you discover how to apply truth tables to a varied array of problems in sentential logic (SL).

Now, imagine that an easier way exists for evaluating SL statements — especially for complicated statements that contain three or more constants. Thinking this way puts you in the right frame of mind for this chapter. A better way beyond plugging and chugging every truth table row for logic problems does exist — the *quick table.* Unlike truth tables, which require you to evaluate a problem under *every* possible interpretation, quick tables use only one row (or two or three at the most!) to do the work of an entire truth table.

In this chapter, you see how quick tables save you time by allowing you to work with SL statements as a *whole* instead of in *parts*, as truth tables do. I show you how to recognize the types of problems that are more easily solved with quick tables than with truth tables. And I walk you through the strategies and methods that help you use quick tables to solve a variety of common logic problems.

Dumping the Truth Table for a New Friend: the Quick Table

Truth tables are ordered, precise, thorough — and tedious! They're tedious because you have to evaluate *every* possible interpretation in order to solve a problem.

REMEMBER

When using truth tables, you start with the *parts* of a statement (the truth values of the constants) and finish with the *whole* statement (the value of the main operator). This method of starting with the parts and ending with the whole is both the strength and the weakness of truth tables. Because you must evaluate a statement under every possible interpretation, you're sure to cover all the bases. For the same reason, however, a lot of your work is repetitive — in other words, tedious and time-consuming.

Considering a new approach to evaluating SL statements

Suppose that, early on Monday morning, your professor walks into class with a cup of coffee, two doughnuts, and the morning paper. They give you an in-class assignment to write up a truth table for the following statement:

$$P \rightarrow ((Q \,\&\, R) \vee (\sim P \,\&\, S))$$

Then, they sit down, open their newspaper, and ignore everybody.

Ugh! With four constants, you're talking 16 rows of evaluation purgatory. But you're a troublemaker, so you go up to the professor's desk and point out that the main operator of the statement is the if-operator, because it's the only operator outside the parentheses. In response, they rattle their newspaper with annoyance.

You persist, carefully explaining your brand-new insight: "Don't you see? All eight interpretations with *P* as false have to make the entire statement true. Isn't that right?" The professor takes a big bite of doughnut and uses their jelly-filled mouth as an excuse not to say anything while they glare at you.

Finally, you get up the courage to ask, "So, how about if I just mark those eight rows as true, without going through all the steps?" They snap, "No!" and you skulk back to your desk.

Like I said, you're a troublemaker. You're probably the type who reads (or maybe even writes) *For Dummies* books.

But you recognize that a lot of truth table interpretations tend to be redundant (unnecessary). So, in many cases, you can eliminate bunches of them all at once. At the same time, however, you need to be careful to throw away only the wrong interpretations and hold onto the right ones. To make sure you're chucking the right interpretations, you need a system called the *quick table*, which I describe throughout this chapter.

Recognizing that a quick table may suffice

REMEMBER

In contrast to truth tables, with quick tables, you start with the *whole* statement — the truth value of the main operator — and finish with the values of its *parts*, which are the values of the constants. The idea behind this method is to start out with just one truth value for a statement and, by making a few smart decisions, save a lot of time by eliminating the repetitive work involved in evaluating every interpretation in a truth table.

You can use quick tables in place of truth tables to test for any of the conditions I discuss in Chapter 6 and list there in Table 6-2.

TIP

Generally speaking, you'll want to toss aside your old familiar truth tables in favor of quick tables for the following three types of problems:

- **Problems that your professor tells you to do using quick tables:** 'nuff said!
- **Problems with four or more constants:** Big truth tables mean big trouble. You're bound to make mistakes no matter how careful you are. In these cases, quick tables are virtually guaranteed to save you time.
- **Problems with statements that are *easy* types:** Some types of statements are easy to crack with quick tables. I show you how to recognize them later in this chapter in the section "Working Smarter (Not Harder) with Quick Tables."

Outlining the Quick Table Process

In this section, I give you an overview of the three basic steps to using quick tables. An example is probably the best way to understand how to use these tables. So, I walk you through an example, and then I fill in more details in later sections.

Each of the three steps to using a quick table can be made easier if you know a few tricks about how to approach it. I provide those a bit later, but for now, just follow along, and if you have any questions, you can find answers later in the chapter.

Here's the example you'll follow through the next couple of sections:

> Suppose you want to know whether the following statements are consistent or inconsistent:
>
> $P \,\&\, Q$ $\qquad$ $Q \rightarrow R$ $\qquad$ $R \rightarrow P$

Making a strategic assumption

All quick tables begin with a *strategic assumption.* In this example, you assume that all three statements are true, and you see where this assumption leads.

Using a strategic assumption, every quick table can lead to two possible outcomes:

- Finding an interpretation under your assumption
- Disproving the assumption by showing that no such interpretation exists

Depending on the problem you're solving, each outcome leads to a different conclusion you can draw.

Think about the example problem this way: If the statements $P \,\&\, Q$, $Q \rightarrow R$, and $R \rightarrow P$ are consistent, the truth values of all three statements are **T** under at least one interpretation (check out Chapter 6 to review the definition of consistency).

> So, a good strategy is to assume that the truth value of each statement is **T** and then see whether you can make this strategic assumption of consistency work. For example, here's what your table would look like:
>
> $P \underset{\mathbf{T}}{\&} Q$ $\qquad$ $Q \underset{\mathbf{T}}{\rightarrow} R$ $\qquad$ $R \underset{\mathbf{T}}{\rightarrow} P$

TIP

Flip to the "Planning Your Strategy" section later in the chapter to see how to set up a quick table to solve every type of problem you've handled with truth tables. There, I provide a complete list of the strategic assumptions you should use for each type of problem.

Filling in a quick table

After the quick table is set up, you look for any further conclusions you can draw about the truth values of any parts of the statement. In the case of this section's ongoing example, because the statement P & Q is true, both sub-statements P and Q are also true.

$$P_{\mathbf{T}} \,\&\, Q \qquad\qquad Q \rightarrow_{\mathbf{T}} R \qquad\qquad R \rightarrow_{\mathbf{T}} P$$

After you know that P and Q are both true, you can fill in this information everywhere these constants appear, like this:

$$P_{\mathbf{T}} \,\&\, Q_{\mathbf{T}} \qquad\qquad Q_{\mathbf{T}} \rightarrow_{\mathbf{T}} R \qquad\qquad R \rightarrow_{\mathbf{T}} P_{\mathbf{T}}$$

Now look at the second statement, $Q \rightarrow R$. The entire statement is true and the first part is also true, so the second part has to be true as well. Therefore, R is true everywhere it appears. So, your table should now look like this:

$$P_{\mathbf{T}} \,\&\, Q_{\mathbf{T}} \qquad\qquad Q_{\mathbf{T}} \rightarrow_{\mathbf{T}} R_{\mathbf{T}} \qquad\qquad R_{\mathbf{T}} \rightarrow_{\mathbf{T}} P_{\mathbf{T}}$$

At this point, all the constants are filled in, so you're ready to read your quick table.

Reading a quick table

When you've completely filled in a quick table, you have a *possible* interpretation, but you need to make sure it really works.

TIP

When you think you have an interpretation that works, check to make sure that

- Every constant has the same truth value everywhere it appears.
- Every evaluation is correct under that interpretation.

The example I'm using passes both tests. Each of the three constants has the same truth value wherever it appears. (For example, the value of P is **T** throughout.) And every evaluation is correct under this interpretation. (For example, the value of P & Q is **T**, which is correct.)

So you've found an interpretation that makes your original assumption correct, which means that all three statements are consistent.

Disproving the assumption

In the ongoing example from the preceding sections, the assumption led to an interpretation. But, as I state in the "Making a strategic assumption" section earlier in the chapter, this doesn't always happen. Sometimes, you may find that an assumption leads to an impossible situation.

For example, suppose you want to know whether the statement $(P \,\&\, Q)$ & $((Q \leftrightarrow R)$ & $\sim P)$ is a *contradiction* — that is, whether its truth value is **F** under every interpretation.

As always, you begin with a strategic assumption. In this case, assume that the statement *isn't* a contradiction, so its truth value is **T** under at least one interpretation.

$$(P \,\&\, Q) \underset{\mathbf{T}}{\&} ((Q \leftrightarrow R) \,\&\, \sim P)$$

As I discuss in Chapter 5, this statement's *main operator* — the only operator that appears outside all the parentheses — is the second and-operator. So this statement is of the form $x \,\&\, y$. Under the assumption that the whole statement is true, you can conclude that both sub-statements are also true, which would make your table look like this:

$$(P \underset{\mathbf{T}}{\&} Q) \underset{\mathbf{T}}{\&} ((Q \leftrightarrow R) \underset{\mathbf{T}}{\&} \sim P)$$

But, notice that both $P \,\&\, Q$ and $(Q \leftrightarrow R) \,\&\, \sim P$ are also of the form $x \,\&\, y$, which means that P, Q, $Q \leftrightarrow R$, and $\sim P$ are all true.

$$(\underset{\mathbf{T}}{P} \underset{\mathbf{T}}{\&} \underset{\mathbf{T}}{Q}) \underset{\mathbf{T}}{\&} ((Q \underset{\mathbf{T}}{\leftrightarrow} R) \underset{\mathbf{T}}{\&} \underset{\mathbf{T}}{\sim} P)$$

So far, so good. In just a couple of steps, you've made a lot of progress. However, here comes trouble: It seems that P and $\sim P$ are both true, which clearly can't be correct. This impossibility disproves the original assumption — which was that the statement *isn't* a contradiction. Therefore, the statement *is* a contradiction.

WARNING

When you disprove an assumption, you need to be careful that you've truly ruled out all possible interpretations. Because the fact that you've ruled them out may not be self-evident from the finished quick table, some professors may require a brief explanation about how you arrived at your conclusion.

Here's the type of explanation that would do just fine for this example:

> Assume that the statement is *not* a contradiction. So, at least one interpretation makes the statement true. Then, $P \,\&\, Q$ and $(Q \leftrightarrow R) \,\&\, \sim P$ are both true. But then, both sub-statements P and $\sim P$ are true, which is impossible; therefore, the statement is a contradiction.

Planning Your Strategy

When testing for any of the conditions listed in Table 6-2 in Chapter 6, you begin with a strategic assumption and then look for an interpretation that fits. If you indeed find such an interpretation, you have one answer; if you find that no fitting interpretation exists, you have another answer.

This section is a summary of the information you need to set up and read a quick table. For each case, I give you the assumption you need to start with, provide an example and your first step, and then tell you how to read both possible outcomes.

In each case, the strategic assumption I give you is the best way (and sometimes the only way) to test for the given condition by using a quick table. That's because the assumption in each case allows you to draw a conclusion based on the existence (or nonexistence) of a *single* interpretation — and quick tables are tailor-made to find a single interpretation if one exists.

Tautology

Strategic assumption: Try to show that the statement is not a tautology, so assume that the statement is false.

Example: Is $(P \rightarrow (Q \rightarrow R)) \rightarrow ((P \,\&\, Q) \rightarrow R)$ a tautology?

First step (assume that the statement is false):

$$(P \rightarrow (Q \rightarrow R)) \underset{\mathbf{F}}{\rightarrow} ((P \,\&\, Q) \rightarrow R)$$

Outcomes:

- **If you find an interpretation under this assumption:** The statement is *not* a tautology — it's either a contradiction or a contingent statement.
- **If you disprove the assumption:** The statement is a tautology.

Contradiction

Strategic assumption: Try to show that the statement is not a contradiction, so assume that the statement is true.

Example: Is $(P \rightarrow (Q \rightarrow R)) \rightarrow ((P \,\&\, Q) \rightarrow R)$ a contradiction?

First step (assume that the statement is true):

$$(P \rightarrow (Q \rightarrow R)) \underset{\mathbf{T}}{\rightarrow} ((P \,\&\, Q) \rightarrow R)$$

Outcomes:

- **If you find an interpretation under this assumption:** The statement is *not* a contradiction — it's either a tautology or a contingent statement.
- **If you disprove the assumption:** The statement is a contradiction.

Contingent statement

Use the previous two tests for tautology and contradiction. If the statement *isn't* a tautology and *isn't* a contradiction, it must be a contingent statement.

Semantic equivalence and inequivalence

Strategic assumption: Try to show that the two statements are semantically inequivalent, so connect them using an if-and-only-if–operator and assume that this new statement is false.

Example: Are $P \,\&\, (Q \vee R)$ and $(P \vee Q) \,\&\, (P \vee R)$ semantically equivalent statements?

First step (assume that the connected statement is false):

$$(P \,\&\, (Q \vee R)) \underset{\mathbf{F}}{\leftrightarrow} ((P \vee Q) \,\&\, (P \vee R))$$

Outcomes:

- **If you find an interpretation under the assumption:** The statements are semantically inequivalent.
- **If you disprove the assumption:** The statements are semantically equivalent.

Consistency and inconsistency

Strategic assumption: Try to show that the set of statements is consistent, so assume that all the statements are true.

Example: Are the statements $P \,\&\, Q$, $\sim(\sim Q \vee R)$, and $\sim R \rightarrow \sim P$ consistent or inconsistent?

First step (assume that all the statements are true):

$$P \underset{\mathbf{T}}{\&} Q \qquad \underset{\mathbf{T}}{\sim}(\sim Q \vee R) \qquad \sim R \underset{\mathbf{T}}{\rightarrow} \sim P$$

Outcomes:

- **If you find an interpretation under the assumption:** The set of statements is consistent.
- **If you disprove the assumption:** The set of statements is inconsistent.

Validity and invalidity

Strategic assumption: Try to show that the argument is invalid, so assume that all the premises are true and the conclusion is false.

Example: Is this argument valid or invalid?

Premises:

$P \rightarrow Q$

$\sim(P \leftrightarrow R)$

Conclusion:

$\sim(\sim Q \,\&\, R)$

First step (assume that all the premises are true and the conclusion is false):

$$P \underset{\mathbf{T}}{\rightarrow} Q \qquad \underset{\mathbf{T}}{\sim}(P \leftrightarrow R) \qquad \underset{\mathbf{F}}{\sim}(\sim Q \,\&\, R)$$

Outcomes:

- **If you find an interpretation under the assumption:** The argument is invalid.
- **If you disprove the assumption:** The argument is valid.

Working Smarter (Not Harder) with Quick Tables

Part of the trade-off in using quick tables is that you have to think about how to proceed instead of just writing out all the possible interpretations of the statement you need to evaluate. So, quick tables become a lot easier when you know which type of statement forms to look for. In this section, I show you how to use quick tables to your best advantage.

In Chapter 5, I discuss the eight basic forms of SL statements as a way to understand evaluation. These forms are even more helpful when you're using quick tables, so take a look at Table 5-1 if you need a quick refresher.

When you're working with quick tables, the truth value of each basic statement form becomes important. Two possible truth values (**T** and **F**) for each of the eight forms gives you 16 different possibilities. Some of these are easier to use in quick tables than others. I start by describing the easy ones in the next section.

Recognizing the six easiest types of statements to work with

Of the 16 types of SL statements (including truth values), six are easy to work with when using quick tables. With each of these types, the truth value of the statement's two sub-statements, *x* and *y*, are simple to figure out.

For example, suppose that you have a statement in the form $x \,\&\, y$ and you know its truth value is **T**. Remember that the only way an and-statement can be true is when both parts of it are true, so you know that the values of both *x* and *y* are also **T**.

Similarly, suppose you have a statement in the form $\sim(x \,\&\, y)$ and you know its truth value is **F**. In this case, it's easy to see that the value of $x \,\&\, y$ is **T**, which again means that the values of both *x* and *y* are **T**.

Figure 7-1 shows you the six easiest types of SL statements to work with. After you recognize these statements, you can often move very quickly through a quick table.

Starting with either:	Leads to:	Values of x and y:
x & y OR ~(x & y) **T** **F**	x & y **T T T**	x is **T** and y is **T**
x ∨ y OR ~(x ∨ y) **F** **T**	x ∨ y **F F F**	x is **F** and y is **F**
x → y OR ~(x → y) **F** **T**	x → y **T F F**	x is **T** and y is **F**

FIGURE 7-1: The six easiest types of SL statements.

For example, suppose you want to find out whether the following is a valid or invalid argument:

Premises:

$$\sim(P \rightarrow (Q \vee R))$$
$$\sim(P \,\&\, (Q \leftrightarrow \sim R))$$

Conclusion:

$$P \,\&\, \sim R$$

The first step is always to pick the correct strategy. In this case, as shown in the "Planning Your Strategy" section earlier in the chapter, you assume the premises to be true and the conclusion to be false. (In other words, you assume that the argument is *invalid*, and you look for an interpretation that fits this assumption.) Your quick table will look like this:

$$\underset{\mathbf{T}}{\sim}(P \rightarrow (Q \vee R)) \qquad \underset{\mathbf{T}}{\sim}(P \,\&\, (Q \leftrightarrow \sim R)) \qquad P \underset{\mathbf{F}}{\&} \sim R$$

Notice that the first statement is in the form $\sim(x \rightarrow y)$, with a truth value of **T**. Referring to Figure 7-1, you know that P is true and $Q \vee R$ is false, which means you can fill in your table like this:

$$\underset{\mathbf{T}}{\sim}(\underset{\mathbf{T}}{P} \underset{\mathbf{F}}{\rightarrow} (\underset{\mathbf{F}}{Q} \underset{\mathbf{F}}{\vee} \underset{\mathbf{F}}{R})) \qquad \underset{\mathbf{T}}{\sim}(P \,\&\, (Q \leftrightarrow \sim R)) \qquad P \underset{\mathbf{F}}{\&} \sim R$$

Now that you know the value of $Q \vee R$ is **F**, you can refer again to Figure 7-1 and see that Q and R are both false.

$$\underset{\mathbf{T}}{\sim}(\underset{\mathbf{T}}{P} \underset{\mathbf{F}}{\rightarrow} (\underset{\mathbf{F}}{Q} \underset{\mathbf{F}}{\vee} \underset{\mathbf{F}}{R})) \qquad \underset{\mathbf{T}}{\sim}(P \,\&\, (Q \leftrightarrow \sim R)) \qquad P \underset{\mathbf{F}}{\&} \sim R$$

In only three steps, you've figured out the truth values of all three constants, so you can fill these in like this:

$$\underset{\mathbf{T}}{\sim}(\underset{\mathbf{T}}{P} \underset{\mathbf{F}}{\rightarrow} (\underset{\mathbf{F}}{Q} \underset{\mathbf{F}}{\vee} \underset{\mathbf{F}}{R})) \qquad \underset{\mathbf{T}}{\sim}(\underset{\mathbf{T}}{P} \underset{\mathbf{F}}{\&} (\underset{\mathbf{F}}{Q} \leftrightarrow \underset{\mathbf{F}}{\sim} \underset{\mathbf{F}}{R})) \qquad \underset{\mathbf{T}}{P} \underset{\mathbf{F}}{\&} \underset{\mathbf{F}}{\sim} \underset{\mathbf{F}}{R}$$

Now, you need to finish filling in the table and check to see whether this interpretation works for every statement.

$$\underset{\mathbf{T}}{\sim}(\underset{\mathbf{T}}{P} \underset{\mathbf{F}}{\rightarrow} (\underset{\mathbf{F}}{Q} \underset{\mathbf{F}}{\vee} \underset{\mathbf{F}}{R})) \qquad \underset{\mathbf{T}}{\sim}(\underset{\mathbf{T}}{P} \underset{\mathbf{F}}{\&} (\underset{\mathbf{F}}{Q} \underset{\mathbf{F}}{\leftrightarrow} \underset{\mathbf{T}}{\sim} \underset{\mathbf{F}}{R})) \qquad \underset{\mathbf{T}}{P} \underset{\mathbf{F}}{\&} \underset{\mathbf{T}}{\sim} \underset{\mathbf{F}}{R}$$

In this case, the second statement is correct. However, the third statement is incorrect: Both parts of the and-statement are true, so the value of the whole statement can't be **F**. This disproves the assumption that the argument is *invalid*, so you know that the argument is *valid*.

Working with the four not-so-easy statement types

REMEMBER

Sometimes, you're going to get stuck with types of SL statements that aren't as easy to work with as the six I introduce in the previous section. This is especially true when you're testing for semantic equivalence because, as I discuss earlier in the chapter in "Planning Your Strategy," the strategy here is to join the two statements with an if-and-only-if–operator.

All four statement types that contain the if-and-only-if–operator give *two* possible sets of values for *x* and *y*, as shown in Figure 7-2.

Starting with either:	Leads to:	Values of x and y:
x ↔ y OR ~(x ↔ y) **T** **F**	x ↔ y **T T T** **F T F**	EITHER x is **T** and y is **T** OR x is **F** and y is **F**
x ↔ y OR ~(x ↔ y) **F** **T**	x ↔ y **T F F** **F F T**	EITHER x is **T** and y is **F** OR x is **F** and y is **T**

FIGURE 7-2: Four not-so-easy types of SL statements.

Suppose you want to find out whether the statements $\sim(P \vee (Q \rightarrow R))$ and $(P \rightarrow R) \,\&\, Q$ are semantically equivalent or inequivalent. The strategy here is to connect the two statements with an if-and-only-if–operator, and then you can assume that this new statement is false. (In other words, assume that the two statements are semantically inequivalent.) Your table would look like this:

$$\sim(P \vee (Q \rightarrow R)) \underset{\mathbf{F}}{\leftrightarrow} ((P \rightarrow R) \;\&\; Q)$$

Refer to Figure 7-2, and note that you can take the following two possible avenues for this statement:

$$\underset{\mathbf{F}}{\overset{\mathbf{T}}{\sim}}(P \vee (Q \rightarrow R)) \underset{\mathbf{F}}{\leftrightarrow} ((P \rightarrow R) \underset{\mathbf{T}}{\overset{\mathbf{F}}{\&}} Q)$$

When looking at the first avenue, notice that the first part of the statement — $\sim(P \vee (Q \rightarrow R))$ — is one of the six easy types to work with — $\sim(x \vee y)$ — so you can fill in your table as follows:

$$\underset{\mathbf{T}}{\sim}(\underset{\mathbf{F}}{P} \underset{\mathbf{F}}{\vee} (Q \underset{\mathbf{F}}{\rightarrow} R)) \underset{\mathbf{F}}{\leftrightarrow} ((P \rightarrow R) \underset{\mathbf{F}}{\&} Q)$$

Furthermore, once you know that $Q \rightarrow R$ is false, you can conclude that Q is true and R is false. Now you can fill in these two values:

$$\underset{\mathbf{T}}{\sim}(\underset{\mathbf{F}}{P} \underset{\mathbf{F}}{\vee} (\underset{\mathbf{T}}{Q} \underset{\mathbf{F}}{\rightarrow} \underset{\mathbf{F}}{R})) \underset{\mathbf{F}}{\leftrightarrow} ((P \rightarrow R) \underset{\mathbf{F}}{\&} Q)$$

After you know the values of all three constants, you can fill in the rest of the table.

$$\underset{\mathbf{T}}{\sim}(\underset{\mathbf{F}}{P} \underset{\mathbf{F}}{\vee} (\underset{\mathbf{T}}{Q} \underset{\mathbf{F}}{\rightarrow} \underset{\mathbf{F}}{R})) \underset{\mathbf{F}}{\leftrightarrow} ((\underset{\mathbf{F}}{P} \underset{\mathbf{T}}{\rightarrow} \underset{\mathbf{F}}{R}) \underset{\mathbf{F}}{\&} \underset{\mathbf{T}}{Q})$$

Under this interpretation, the two parts of the and-statement are true, but the statement itself is false, which is incorrect. So, the search for an interpretation that works goes on.

Now, you can try the second avenue.

$$\underset{\mathbf{F}}{\sim}(P \vee (Q \rightarrow R)) \underset{\mathbf{F}}{\leftrightarrow} ((P \rightarrow R) \underset{\mathbf{T}}{\&} Q)$$

In this avenue, the second part of the statement — $(P \rightarrow R) \,\&\, Q$ — is an easy type — $x \,\&\, y$ — so you can fill in the table as follows:

$$\underset{\mathbf{F}}{\sim}(P \vee (Q \rightarrow R)) \underset{\mathbf{F}}{\leftrightarrow} ((P \underset{\mathbf{T}}{\rightarrow} R) \underset{\mathbf{T}}{\&} \underset{\mathbf{T}}{Q})$$

Now, you know that Q is true. Also, the or-statement in the first part is true because its negation is false. So, you fill in the table, which should now look like this:

$$\underset{\mathbf{F}}{\sim}(P \underset{\mathbf{T}}{\vee} (\underset{\mathbf{T}}{Q} \rightarrow R)) \underset{\mathbf{F}}{\leftrightarrow} ((P \underset{\mathbf{T}}{\rightarrow} R) \underset{\mathbf{T}}{\&} \underset{\mathbf{T}}{Q})$$

At this point, you have no more solid conclusions to make. But you're close to being finished, and with a quick table you need to find only one interpretation that works. In this case, I suggest that you make a guess and see where it takes you.

Suppose, for example, that the value of P is **T**. Then, to make the sub- statement $P \rightarrow R$ true, the value of R would also have to be **T**. So, it looks like you get a perfectly good interpretation of the statement when the values of all three constants are **T**. Fill in your table so that it looks like this:

$$\underset{\mathbf{F}}{\sim}(\underset{\mathbf{T}}{P} \underset{\mathbf{T}}{\vee} (\underset{\mathbf{T}}{Q} \underset{\mathbf{T}}{\rightarrow} \underset{\mathbf{T}}{R})) \underset{\mathbf{F}}{\leftrightarrow} ((\underset{\mathbf{T}}{P} \underset{\mathbf{T}}{\rightarrow} \underset{\mathbf{T}}{R}) \underset{\mathbf{T}}{\&} \underset{\mathbf{T}}{Q})$$

This checks out because, as I discuss in "Reading a quick table," every constant has the same truth value everywhere it appears, and the entire evaluation is correct under that interpretation. So, you've found an interpretation under your original assumption, which means that the two statements are semantically inequivalent.

TECHNICAL STUFF

You may be wondering what would have happened if you had guessed that the value of P is **F**. In this case, you would have found an alternative interpretation, with the truth value of R being **T**. But it doesn't matter — with quick tables, you need to find only one interpretation, and you're done.

Coping with the six difficult statement types

Six types of SL statements don't lend themselves very well to quick tables. Figure 7-3 shows why this is so.

The table shows that each of these types of statements leads to three possible avenues, and if you're using a quick table to solve this problem, you're in for a long, cumbersome search. Fortunately, you have other options. The following sections show you the options when you're facing three possible outcomes.

Starting with either:	Leads to:	Values of x and y:
x & y OR ~(x & y) **F** **T**	x & y **T F F** **F F T** **F F F**	EITHER x is **T** and y is **F** OR x is **F** and y is **T** OR x is **F** and y is **F**
x ∨ y OR ~(x ∨ y) **T** **F**	x ∨ y **T T T** **T T F** **F T T**	EITHER x is **T** and y is **T** OR x is **T** and y is **F** OR x is **F** and y is **T**
x → y OR ~(x → y) **T** **F**	x → y **T T T** **F T T** **F T F**	EITHER x is **T** and y is **T** OR x is **F** and y is **T** OR x is **F** and y is **F**

FIGURE 7-3: The six difficult types of SL statements.

The first way: Use a truth table

One way to avoid the trouble of searching through the three avenues for these types of statements is to fall back on truth tables. Truth tables are always an option (unless your professor forbids them), and they'll always give you the right answer. The downside of truth tables, as always, is that if a problem has a lot of different constants, you're in for a lot of work.

The second way: Use a quick table

The second way to solve a difficult problem is just to grit your teeth and try all three avenues with a quick table. After all, you figured out how to chase down two avenues in the previous example. And besides, you may get lucky and find an interpretation that works on the first try.

For example, suppose you want to find out whether the statement $(\sim(P \rightarrow Q) \,\&\, \sim(R \vee S)) \rightarrow (Q \leftrightarrow R)$ is a contradiction. The strategy in this case is to make the following assumption:

$$(\sim(P \rightarrow Q) \,\&\, \sim(R \vee S)) \underset{\mathbf{T}}{\rightarrow} (Q \leftrightarrow R)$$

which leads to these three possible avenues:

$$(\sim(P \rightarrow Q) \underset{\substack{\mathbf{T}\\ \mathbf{F}\\ \mathbf{F}}}{\&} \sim(R \vee S)) \underset{\mathbf{T}}{\rightarrow} (Q \underset{\substack{\mathbf{T}\\ \mathbf{T}\\ \mathbf{F}}}{\leftrightarrow} R)$$

Fortunately, the first avenue leads someplace promising: Notice that the sub-statement $\sim(P \rightarrow Q) \,\&\, \sim(R \vee S)$ is an and-statement whose value is **T**, which is one of the six easy types of statements. The table looks like this:

$$(\underset{\mathbf{T}}{\sim}(P \rightarrow Q) \underset{\mathbf{T}}{\&} \underset{\mathbf{T}}{\sim}(R \vee S)) \underset{\mathbf{T}}{\rightarrow} (Q \underset{\mathbf{T}}{\leftrightarrow} R)$$

Even better, the two smaller statements are also easy types, leaving the table looking like this:

$$(\underset{\mathbf{T}}{\sim}(\underset{\mathbf{T}}{P} \underset{\mathbf{F}}{\rightarrow} \underset{\mathbf{F}}{Q}) \underset{\mathbf{T}}{\&} \underset{\mathbf{T}}{\sim}(\underset{\mathbf{F}}{R} \underset{\mathbf{F}}{\vee} \underset{\mathbf{F}}{S})) \underset{\mathbf{T}}{\rightarrow} (Q \underset{\mathbf{T}}{\leftrightarrow} R)$$

Now you can plug in the values of Q and R.

$$(\underset{\mathbf{T}}{\sim}(\underset{\mathbf{T}}{P} \underset{\mathbf{F}}{\rightarrow} \underset{\mathbf{F}}{Q}) \underset{\mathbf{T}}{\&} \underset{\mathbf{T}}{\sim}(\underset{\mathbf{F}}{R} \underset{\mathbf{F}}{\vee} \underset{\mathbf{F}}{S})) \underset{\mathbf{T}}{\rightarrow} (\underset{\mathbf{F}}{Q} \underset{\mathbf{T}}{\leftrightarrow} \underset{\mathbf{F}}{R})$$

In just three steps, you found an interpretation that works under the assumption that the statement is true, so the statement is not a contradiction.

REMEMBER

You won't always get lucky enough to find an interpretation that works when exploring the first avenue, but in this case, a quick table still worked a lot faster than a truth table. Good luck!

The third way: Use a truth tree

In Chapter 8, I introduce *truth trees* as a perfect alternative for situations like these. When truth tables are too long and quick tables are too difficult, a truth tree may be your best friend.

IN THIS CHAPTER

- Decomposing SL statements with truth trees
- Testing for consistency, validity, and semantic equivalence
- Categorizing tautologies, contradictions, and contingent statements with truth trees

Chapter 8
Knowing How Truth Grows on Trees

In Chapters 6 and 7, I show you how to use truth tables and quick tables to solve problems in sentential logic (SL) statements and sets of statements. In this chapter, I introduce you to the third (and my favorite!) method for solving logic problems: truth trees.

In the following sections, I show you how truth trees work by a process of decomposing SL statements into their sub-statements. Then I show you how to use truth trees to solve the same types of problems you can handle by using truth tables and quick tables.

Understanding How Truth Trees Work

Truth trees are a powerful tool for determining information about SL statements. In fact, I think they're the best tool in this book to solve nearly every type of problem you'll come across in logic. Why? Glad you asked. Here's the answer:

- **Truth trees are easy.** They're easy to learn and easy to use.

REMEMBER

Truth trees combine the best features of truth tables and quick tables, but without the baggage of either.

- *Like truth tables*, truth trees are a plug-and-chug method (although a couple of smart choices along the way can still help you out), but they're much shorter.
- *Like quick tables*, truth trees avoid repetitive evaluation, but they never lead to trouble such as guesswork or problems with three possible avenues to eliminate.

» **Truth trees are a perfect way to solve SL problems of *any* size with maximum efficiency.** They're also useful in quantification logic (QL), which is the larger logical system I discuss in Part 4.

In Chapter 5, I discuss the eight basic forms of SL statements. These become important again as you study truth trees, because truth trees handle each of these forms in a different way. (If you need a refresher on the eight forms of SL statements, refer to Table 5-1.)

Decomposing SL statements

Truth trees work by *decomposing* statements — that is, breaking them into smaller sub-statements. And you then draw conclusions based on the truth value of these sub-statements.

Consider these examples:

» If you know that the statement $(P \vee Q)\ \ \&\ \ (Q \vee R)$ is true, you know that both the sub-statement $(P \vee Q)$ is true and the sub-statement $(Q \vee R)$ is true. In general, then, you can break any true statement of the form *x* & *y* into two true statements, *x* and *y*.

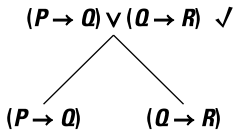
» If you know that the statement $(P \rightarrow Q) \vee (Q \leftrightarrow R)$ is true, you know that either the sub-statement $(P \rightarrow Q)$ or the sub-statement $(Q \leftrightarrow R)$ is true. In general, then, you can break any true statement of the form *x* ∨ *y* into two statements, *x* and *y*, at least one of which is true.

REMEMBER

When you use truth trees, forms of true statements that lead directly to other true statements are called *single-branching* statements. Forms that lead in two possible directions (true or false) are called *double-branching* statements. Double-branching statements always lead to either one or two sub-statements per branch.

Figure 8-1 shows a list of all eight basic forms of SL statements with their decompositions.

For example, if you're decomposing the statement $(P \to Q) \vee (Q \to R)$, you begin by breaking it into two sub-statements as follows:



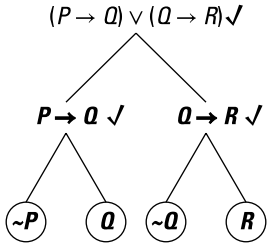
Notice that you place a check mark next to the statement $(P \to Q) \vee (Q \to R)$ after you decompose it into its two sub-statements. The check mark lets you know that you're finished with the associated statement — and now need to deal with its two sub-statements.

Next, you break each sub-statement into smaller sub-statements, according to the rules provided in Figure 8-1.

Single Branching	***Double Branching to Two Sub-Statements***	***Double Branching to One Sub-Statement***
$x \,\&\, y$ x y	$x \leftrightarrow y$ x $\quad$ $\sim x$ y $\quad$ $\sim y$	$\sim(x \,\&\, y)$ $\sim x$ $\quad$ $\sim y$
$\sim(x \vee y)$ $\sim x$ $\sim y$	$\sim(x \leftrightarrow y)$ x $\quad$ $\sim x$ $\sim y$ $\quad$ y	$x \vee y$ x $\quad$ y
$\sim(x \to y)$ x $\sim y$		$x \to y$ $\sim x$ $\quad$ y

FIGURE 8-1: The eight types of SL statements with decompositions.

The completed decompositions show how truth trees got their name. The final structure resembles an upside-down tree, like this:



In this example, you again check the statements that you decompose. Also note the convention of circling the end results. After you decompose a statement down to either a single constant or its negation, circling makes keeping track of the result easier. In this case, you circle ~*P*, *Q*, ~*Q*, and *R*.

REMEMBER

After you break down every statement and sub-statement in this way, the truth tree is complete. Each branch tells you something about one or more interpretations that would make the original statement true. To find these interpretations, trace from the beginning of the trunk all the way to the end of a branch and note all the circled statements you pass through along the way.

For example, tracing from the beginning of the trunk to the end of the first branch, the only circled statement you pass through is ~*P*. So, this branch tells you that *any* interpretation in which *P* is false will make the original statement true.

Solving problems with truth trees

REMEMBER

You can use a truth tree to solve any problem that you can also solve by using a truth table or a quick table. As with these other tools, truth trees follow a step-by-step process. Here are the steps for solving a logic problem by using a truth tree:

1. **Set up the truth tree trunk.**

   The *trunk* consists of the statement or statements you need to decompose, and you construct it according to the type of problem you're trying to solve.

2. **Fill in the truth tree branches.**

   To fill in a truth tree, use the rules of decomposition that you can find in Figure 8-1 to create all of its branches.

3. **Read the completed truth tree to determine the outcome.**

   You can find out which of the following two outcomes has occurred:

   - **At least one branch is left open:** At least one interpretation makes every statement in the trunk of the tree true.
   - **All the branches are closed:** No interpretation makes every statement in the trunk of the tree true. (In the following section, I show you how to close off a branch.)

Showing Consistency or Inconsistency

You can use truth trees to figure out whether a set of statements is consistent or inconsistent. (See Chapter 6 for more information about consistency.) For example, suppose you want to figure out whether the three statements $P \,\&\, {\sim}Q$, $Q \vee {\sim}R$, and ${\sim}P \rightarrow R$ are consistent or inconsistent.

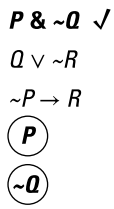
Constructing the truth tree

REMEMBER

To decide whether a set of statements is *consistent* (at least one interpretation makes all the statements true) or *inconsistent* (no interpretation makes all of them true), construct a truth tree using that set of statements as its trunk. Here's your trunk:

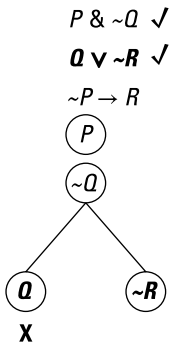
$$P \,\&\, {\sim}Q$$
$$Q \vee {\sim}R$$
$${\sim}P \rightarrow R$$

After you create your trunk, you can begin decomposing the first statement, $P \,\&\, {\sim}Q$. Here's what you get:



Check the statement $P \,\&\, {\sim}Q$ after you decompose it into its two sub-statements and circle the single constants, P and ${\sim}Q$.

The next statement is $Q \vee {\sim}R$, which decomposes along two separate branches as follows:



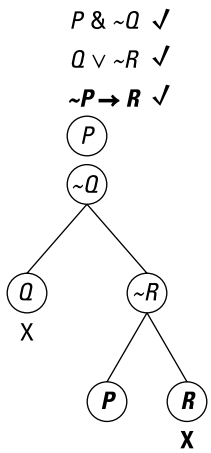
In this example, tracing all the way from the beginning of the trunk to the end of the branch on the left forces you to pass through the three circled statements P, ${\sim}Q$, and Q. But ${\sim}Q$ and Q are contradictory statements, so you can close off that branch by marking it with an X at the bottom.

REMEMBER

When you find that tracing from the beginning of a trunk to the end of a branch forces you to pass through a pair of contradictory circled statements, close off that branch with an X.

Closing off this branch makes sense when you think about it. The branch tells you that any interpretation in which the statements P, ${\sim}Q$, and Q are true would make all three original statements true. But, ${\sim}Q$ and Q can't both be true, so this branch provides *no* possible interpretations.

The final statement to decompose is $\sim P \rightarrow R$. As you can see by referring to Figure 8-1, the branches for this statement look like this:



Notice that you close off the branch in which you reach another contradiction. In this case, the contradiction is R and $\sim R$ (which cannot both be true).

REMEMBER

Your truth tree is finished when one of the following occurs:

- Every statement or constant has been either checked or circled.
- Every branch has been closed off.

Determining consistency or inconsistency from the truth tree

In the example problem from the previous section, every statement or constant has been either checked or circled, so the tree is finished. After the tree is finished, you can check to see whether any branches are still open. The presence or absence of open branches on a finished truth tree enables you to determine whether the original set of statements is consistent or inconsistent. Follow these guidelines:

- If the finished truth tree has at least one open branch, the set of statements is *consistent*. (That is, an evaluation exists where all of the statements are true.)
- If the finished truth tree has all closed branches, the set of statements is *inconsistent*.

The previous section's example still has one branch open, which means an interpretation exists under which the three statements are all true. So, in this case, the set of statements is consistent.

If you want to know which interpretation makes all three statements true, just trace from the trunk to the end of the still-open branch. When you trace the length of this branch, you find that the circled statements are P, $\sim Q$, and $\sim R$ and the final P. So, the only interpretation that makes the three original statements true happens when the value of P is **T** and the values of both Q and R are **F** — in which case, the values of both ~Q and ~R are **T**.

Testing for Validity or Invalidity

Truth trees are also handy when you want to determine an argument's validity or invalidity (see Chapter 6 for more information about validity). For example, suppose that you want to figure out whether the following argument is valid or invalid:

Premises:

$$\sim P \leftrightarrow Q$$

$$\sim (P \vee R)$$

Conclusion:

$$\sim Q \,\&\, \sim R$$

REMEMBER

To decide whether an argument is valid or invalid, construct a truth tree using the premises and the *negation* of the conclusion as its trunk.

Using the example I introduced at the beginning of this section, create a trunk that looks like this:

$$\sim P \leftrightarrow Q$$
$$\sim (P \vee R)$$
$$\sim (\sim Q \,\&\, \sim R)$$

Choosing the order for decomposing

TIP

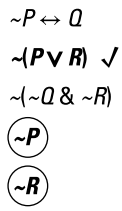
You don't have to start at the top of the tree, however. Decomposing statements in a different order is often helpful. Figure 8-1 divides the eight basic statement forms into three columns. This division can help you decide the order to use for decomposing your statements. When you have more than one statement in the trunk of your truth tree, decompose them in this order:

- Single branching first
- Double branching with two sub-statements next
- Double branching with one sub-statement last

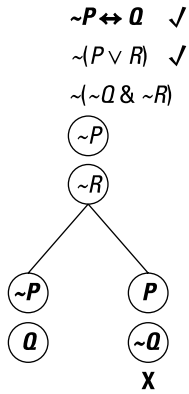
Decomposing statements in this order makes sense. Whenever you can, choose a single branch path to keep your trees as small as possible. But, when you have no choice but to double branch, choose a double branch with two sub-statements. Adding two statements increases the chance that you'll be able to close off one of the branches.

Following the order

In the example from the previous section, only the second statement, $\sim(P \vee R)$, leads to a single branch. So, you decompose it first, like this:

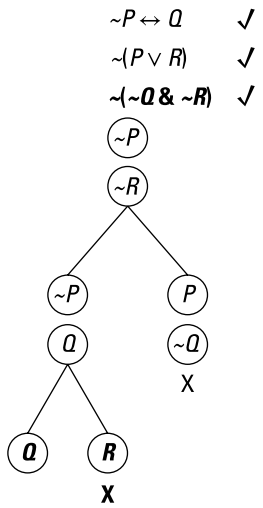


Both of the remaining statements lead to double branches. But only the first statement, $\sim P \leftrightarrow Q$, decomposes to two sub-statements. So, you decompose that one next.



At this point, one branch is closed off: Tracing from the beginning of the trunk to the end of this branch forces you to pass through both $\sim P$ and P, which is a contradiction.

The final step, decomposing $\sim(\sim Q \,\&\, \sim R)$, leads your truth tree to look like this:



Notice that you add the new decomposition only to the open branch, but not to the closed-off branch. Now every statement is either checked or circled, so the tree is complete.

Checking the validity or invalidity

REMEMBER

When you're checking a truth tree for validity or invalidity, the following guidelines apply:

- If the truth tree has at least one open branch, the argument is *invalid*. (That is, an evaluation exists where all the premises are true and the conclusion is false.)
- If the truth tree has all closed branches, the argument is *valid*.

In this section's example, one branch is still open, so the argument is invalid. Trace from the trunk to the end of this still-open branch and you will find the circled statements $\sim P$, Q, and $\sim R$. This tells you that the *only* interpretation under which this argument is invalid is when the values of both P and R are **F** and the value of Q is **T.**

Separating Tautologies, Contradictions, and Contingent Statements

In Chapter 6, I show you that every statement in SL is categorized as a *tautology* (a statement that's always true), a *contradiction* (a statement that's always false), or a *contingent statement* (a statement that can be either true or false, depending on the value of its constants). You can use truth trees to separate SL statements into these three categories.

Tautologies

Suppose you want to test the statement $((P \& Q) \vee R) \rightarrow ((P \leftrightarrow Q) \vee (R \vee (P \ \& \sim Q)))$ to determine whether it's a tautology.

REMEMBER

To show that a statement is a tautology, construct a truth tree using the *negation* of that statement as its trunk.

Using the negation of the example statement I just introduced, you can create a trunk that looks like this:

$$\sim(((P \& Q) \vee R) \rightarrow ((P \leftrightarrow Q) \vee (R \vee (P \& \sim Q))))$$

Decomposing based on basic statement forms

Even though this statement looks big and hairy, you know that it corresponds to one of the eight basic forms from Chapter 5 (listed in Table 5-1.) You just need to figure out which one. I go into this in Chapter 5, but a little refresher here wouldn't hurt.

The main operator is the first not-operator (~) — the only operator that is outside all the parentheses — so the statement is one of the four negative forms. And the scope of the if-operator ($\rightarrow$) covers the rest of the statement, so this statement is of the form $\sim(x \rightarrow y)$. As you determine by referring to Figure 8-1, this form is single branching, so your truth tree should now look like this:

$\sim(((P \& Q) \vee R) \rightarrow ((P \leftrightarrow Q) \vee (R \vee (P \& \sim Q)))) \ \checkmark$
$(P \& Q) \vee R$
$\sim((P \leftrightarrow Q) \vee (R \vee (P \& \sim Q)))$

TECHNICAL STUFF

Notice that I remove the outer parentheses from the sub-statement $((P \& Q) \vee R)$. This is a legitimate step, as I explain in Chapter 14.

Now, the last statement is of the form $\sim(x \vee y)$, which is also single branching. Therefore, according to the order in which you should decompose multiple statements (see the "Choosing the order for decomposing" section earlier in the chapter), that's the statement to work on next. Fill in your truth tree as follows:

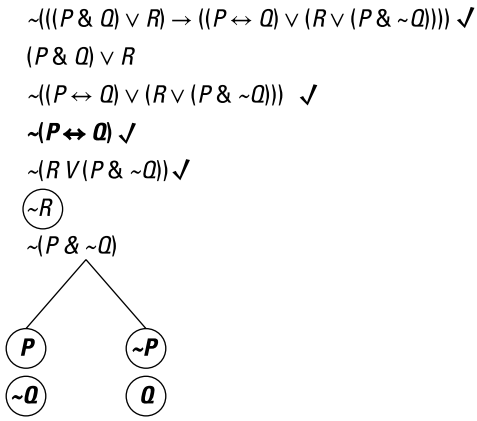
$\sim(((P \& Q) \vee R) \rightarrow ((P \leftrightarrow Q) \vee (R \vee (P \& \sim Q))))$ ✓
$(P \& Q) \vee R$
$\sim((P \leftrightarrow Q) \vee (R \vee (P \& \sim Q)))$ ✓
$\sim(P \leftrightarrow Q)$
$\sim(R \vee (P \& \sim Q))$

Again, the last statement is of the form $\sim(x \vee y)$, so it's single branching, and therefore next up. Here's what your truth tree should look like:

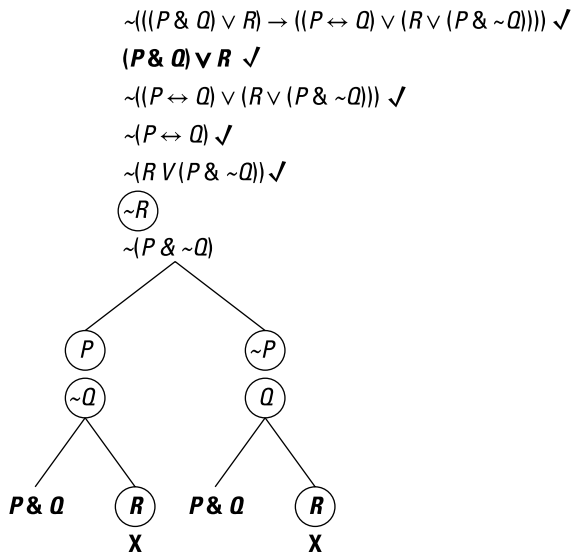
$\sim(((P \& Q) \vee R) \rightarrow ((P \leftrightarrow Q) \vee (R \vee (P \& \sim Q))))$ ✓
$(P \& Q) \vee R$
$\sim((P \leftrightarrow Q) \vee (R \vee (P \& \sim Q)))$ ✓
$\sim(P \leftrightarrow Q)$
$\sim(R \vee (P \& \sim Q))$ ✓
$\sim R$
$\sim(P \& \sim Q)$

Even though this example may look long, step back a minute and notice that you've already taken three steps without double branching. Not having to double branch saves you tons of work as you proceed because you need to worry about only one branch rather than two (or four, or eight!).

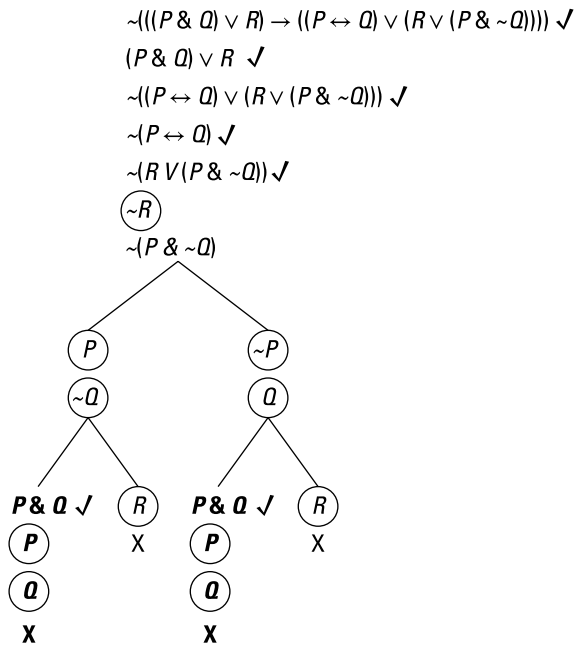
Eventually, though, you have to double branch with this example. Because it branches to two sub-statements, start with the statement $\sim(P \leftrightarrow Q)$, according to the order of decomposition I provide in the earlier section "Choosing the order for decomposing." Your truth tree should now look like this:



Now decompose $(P \,\&\, Q) \vee R$:



This step closes off two of the four branches. Now, $P \& Q$ is single branching, so decompose this statement on both remaining branches, like this:



You've now closed all the remaining branches. Note that the statement $\sim(P \& \sim Q)$ is *not* checked. This makes no difference, though, because after every branch is closed off, the tree is finished.

Checking for a tautology

REMEMBER

When testing to determine whether a statement is a tautology, follow these guidelines:

- If the truth tree has at least one open branch, the statement is *not* a tautology — it's either a contradiction or a contingent statement. (To confirm or rule out contradiction as a possibility, you would need another tree, as I show you in the next section.)
- If the truth tree has no open branches, the statement is a tautology. (That is, no evaluation exists where the *negation* of that statement is true.)

In this section's example, the tree shows you that the statement is a tautology.

Contradictions

Suppose you want to test the statement $(P \leftrightarrow Q)\ \&\ (\sim(P \& R)\ \&\ (Q \leftrightarrow R))$ to determine whether it's a contradiction.

REMEMBER

To show that a statement is a contradiction, construct a truth tree using the statement as its trunk. The trunk for the example I introduce in this section looks like this:

$$(P \leftrightarrow Q)\ \&\ (\sim(P \& R)\ \&\ (Q \leftrightarrow R))$$

Decomposing with single and double branching

Fortunately, the first decomposition of this statement is single branching.

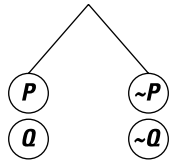
$$(P \leftrightarrow Q)\ \&\ (\sim(P \& R)\ \&\ (Q \leftrightarrow R))\ \checkmark$$
$$P \leftrightarrow Q$$
$$\sim(P \& R)\ \&\ (Q \leftrightarrow R)$$

Now you have a choice to decompose either $P \leftrightarrow Q$ or $\sim(P \& R)\ \&\ (Q \leftrightarrow R)$. But, according to Figure 8-1, the last statement is single branching, so take that route first. Your truth tree should now look like this:

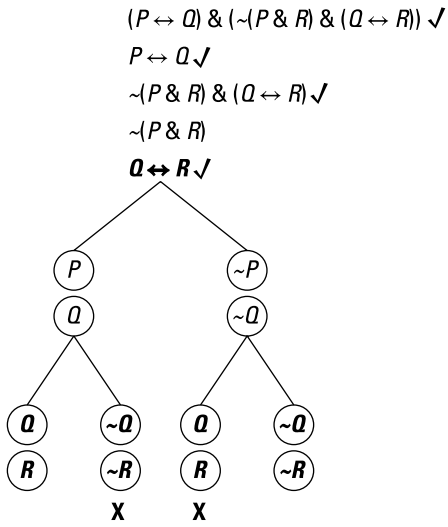
$$(P \leftrightarrow Q)\ \&\ (\sim(P \& R)\ \&\ (Q \leftrightarrow R))\ \checkmark$$
$$P \leftrightarrow Q$$
$$\sim(P \& R)\ \&\ (Q \leftrightarrow R)\ \checkmark$$
$$\sim(P \& R)$$
$$Q \leftrightarrow R$$

Because you've run out of single-branching statements, it's time to double branch, starting with statements that produce two sub-statements. Begin with $P \leftrightarrow Q$ as follows:

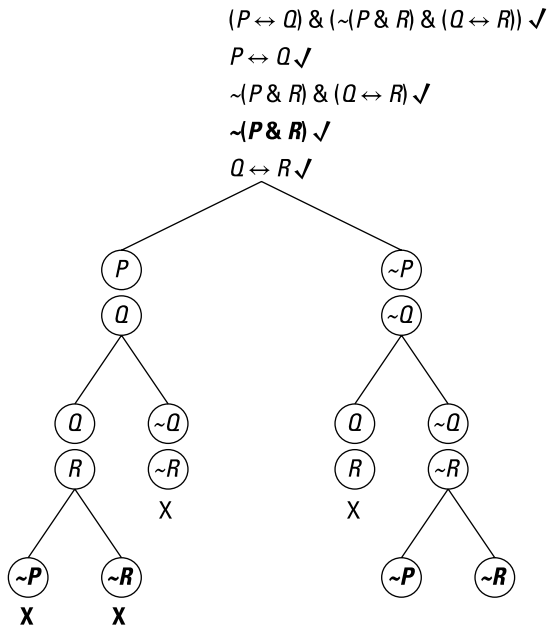
$$(P \leftrightarrow Q)\ \&\ (\sim(P \& R)\ \&\ (Q \leftrightarrow R))\ \checkmark$$
$$P \leftrightarrow Q\ \checkmark$$
$$\sim(P \& R)\ \&\ (Q \leftrightarrow R)\ \checkmark$$
$$\sim(P \& R)$$
$$Q \leftrightarrow R$$



And now decompose $Q \leftrightarrow R$ like this:



This step closes off two of the four branches. Your last step is to decompose ~ (P & R) like this:



Because every statement is either checked or circled, the tree is now complete.

Testing for a contradiction

REMEMBER

When testing to determine whether a statement is a contradiction, apply these guidelines:

- If the truth tree has at least one open branch, the statement is *not* a contradiction — it's either a tautology or a contingent statement. (To confirm or rule out tautology as a possibility, you would need another tree, as I show in the previous section.)
- If the truth tree has no open branches, the statement is a contradiction. (That is, no evaluation exists where that statement is true.)

In this example, two branches remain open, so the statement isn't a contradiction.

Even though two branches remain open in this tree, only one interpretation makes the original statement true. Trace from the trunk to the end of either open branch to see that this interpretation is that P, Q, and R are all false.

Contingent statements

Checking to see whether a statement is contingent is, as always, just a matter of ruling out that it's either a tautology or a contradiction. (Flip to Chapter 6 for details on contingent statements.)

When testing to determine whether a statement is contingent, use the previous two tests for tautology and contradiction. If the statement *isn't* a tautology and *isn't* a contradiction, it must be a contingent statement.

Checking for Semantic Equivalence or Inequivalence

If you have to check a pair of statements for semantic equivalence or inequivalence, you're in luck, because truth trees can help you out. (As I explain in Chapter 6, when two statements are *semantically equivalent*, they both have the same truth value under every interpretation.)

REMEMBER

To decide whether a pair of statements is semantically equivalent or inequivalent, you have to construct two truth trees as follows:

- One tree using the first statement and the negation of the second statement as its trunk
- The other tree using the negation of the first statement and the second statement as its trunk

Suppose you want to find out whether the statements $\sim P \rightarrow (Q \rightarrow \sim R)$ and $\sim(P \vee \sim Q) \rightarrow \sim R$ are semantically equivalent or inequivalent. In this case, you need to make two trees, with trunks as follows:

Tree #1:	Tree #2:
$\sim P \rightarrow (Q \rightarrow \sim R)$	$\sim(\sim P \rightarrow (Q \rightarrow \sim R))$
$\sim(\sim(P \vee \sim Q) \rightarrow \sim R)$	$\sim(P \vee \sim Q) \rightarrow \sim R$

Notice that the first trunk includes the first statement and the negation of the second, while the second trunk includes the negation of the first statement and the second. This is the typical setup for checking semantic equivalence.

Creating the first truth tree

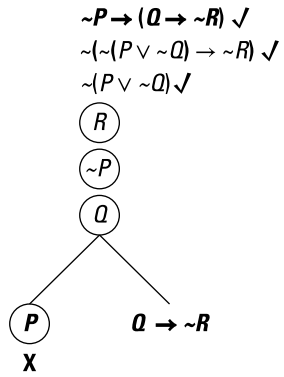
Starting with Tree #1, the second statement is of the single-branching form $\sim(x \rightarrow y)$, so decompose this statement first.

$\sim P \rightarrow (Q \rightarrow \sim R)$
$\sim(\sim(P \vee \sim Q) \rightarrow \sim R)$ ✓
$\sim(P \vee \sim Q)$
R

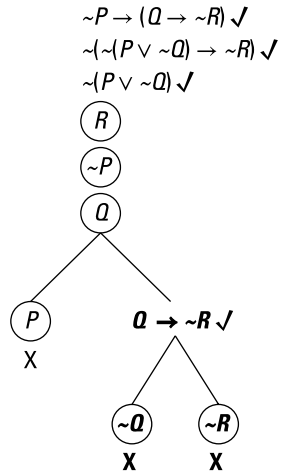
The statement $\sim(P \vee \sim Q)$ is also single branching, so work on it next.

$\sim P \rightarrow (Q \rightarrow \sim R)$
$\sim(\sim(P \vee \sim Q) \rightarrow \sim R)$ ✓
$\sim(P \vee \sim Q)$ ✓
R
$\sim P$
Q

Next, move on to the first statement $\sim P \rightarrow (Q \rightarrow \sim R)$, which you skipped initially.



This step closes off one branch. Now, decomposing $Q \rightarrow \sim R$, you get the following:



Tree #1 is now complete.

REMEMBER

When testing a pair of statements to determine whether they're semantically equivalent or inequivalent, follow these guidelines:

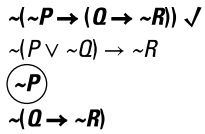
- If *either* truth tree has at least one open branch, the statements are *semantically inequivalent*. (That is, it's possible that both one statement and the *negation* of the other are both true.)
- If *both* truth trees have all closed branches, the statements are *semantically equivalent*.

TIP

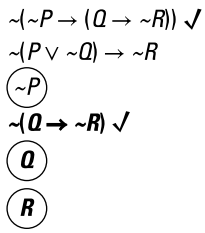
If the first tree has at least one open branch, the statements are *semantically inequivalent,* which means you can skip the second tree.

Creating the second tree (if you need to)

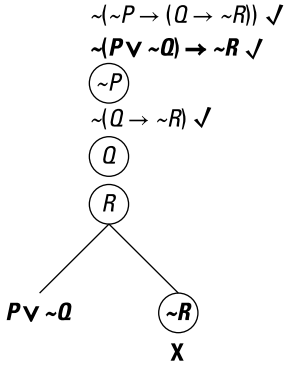
Because the first tree in the previous section's example has all closed branches, you need to move on to the second tree. In this case, the first statement is single branching, so decompose it first as follows:



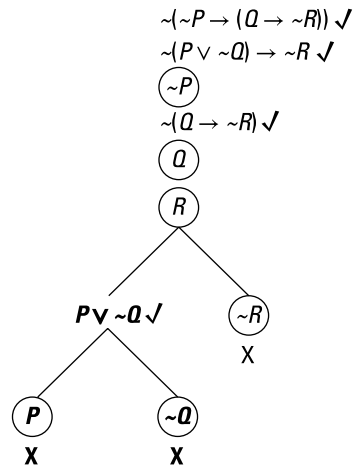
The statement $\sim(Q \rightarrow \sim R)$ is also single branching, so work on it next.



The remaining statement, $\sim(P \vee \sim Q) \rightarrow \sim R$, is double branching, so decompose it next, like this:



This step closes off one branch. Finally, decomposing $P \vee {\sim}Q$ gives you this:



All the branches on each tree are closed off. Thus, because all the branches are closed off *on both trees*, the statements are semantically equivalent.

3

Exploring Proofs, Syntax, and Semantics in SL

IN THIS PART . . .

Discover what a proof looks like and how to construct one by using the first eight rules of inference, which are the implication rules.

Work with the remaining ten rules of inference, which make up the set of equivalence rules.

Add two new proof methods — conditional proofs and indirect proofs — to your logic arsenal.

See the big strategic picture of how and when to use the various logic tools.

Understand the five SL operators on a granular level.

Know the distinction between the syntax and semantics of SL, including the basics of well-formed formulas and Boolean algebra.

IN THIS CHAPTER

- Introducing formal direct proofs
- Constructing proofs using the implication rules of inference
- Understanding the if, and, or, and double-if rules

Chapter 9
Providing Proof for What You Propose

You may already have some experience writing proofs, like those tricky problems that were an important part of high school geometry. In geometry proofs, you started with a set of simple *axioms* (or rules, also called *postulates*), such as "All right angles are equal," and figured out how to build toward more complex statements called *theorems*.

Computer programming, in which you use simple statements to create complex software, also resembles the proof method. And the idea of complexity arising out of simplicity is common to proofs in sentential logic (SL) as well.

In a sense, constructing a proof is like building a bridge from one side of a river to the other. The starting point is the set of premises you're given, and the end point is the conclusion you're trying to reach. The pieces you use to build the bridge are the *rules of inference*, a set of 18 ways to turn old statements into new ones.

In this chapter, I introduce you to the first eight rules of inference — the set of *implication rules*. And along the way, you find out a bit about what proofs look like.

REMEMBER

Even though these rules are clear and unambiguous, it's not always obvious how to use them in a particular case. Figuring out the art to writing proofs can make the process interesting and satisfying when you get it right, but frustrating when you don't. The good news is that a bunch of great tricks exist, and once you know them, you have a lot of ammo to use when the going gets tough.

Bridging the Premises-Conclusion Divide

A valid argument is like a bridge — it gives you a way to get from here (the premises) to there (the conclusion), even if rough waters are rushing below. Proofs give you a way to build a bridge you can trust to get you safely to the other side.

For example, take the following argument:

Premises:

$P \rightarrow Q$

P

Conclusion:

Q

Because proofs concentrate so heavily on arguments, in this chapter I introduce a new space-saving convention for writing arguments. Using this convention, you can write the preceding argument as

$$P \rightarrow Q,\ P :\ Q$$

As you can see, a comma separates the premises ($P \rightarrow Q$ and P). Then a colon separates the premises from the conclusion (Q). When you say that an argument is valid, you're saying that your bridge from premises to conclusion is safe. If you start on the left side (with the premises being true), you're safe to cross over to the right side (meaning the conclusion is also true).

For example, here's an argument in English with the same form as the P and Q argument I show earlier:

Premises:

If we want to go to Koh Samui, then we need to buy plane tickets.

We want to go to Koh Samui.

Conclusion:

We need to buy plane tickets.

REMEMBER

After you know that the premises of a valid argument are true, you know that the conclusion is also true. And it doesn't matter which particular premises you fill in. As long as you start out with true premises and keep the general form of the argument, you always end up with a true conclusion.

Understanding this point (that a set of true premises always leads to a true conclusion) is essential for understanding proofs in SL. For the rest of this chapter, and in Chapters 10 through 12, you discover how to formally prove logical arguments by using one or more premises to demonstrate the conclusion.

Engaging the Eight Implication Rules in SL

Proofs in SL work very much like proofs in geometry, which you may remember from high school. The only difference is that instead of using geometric symbols for shapes and angles, they use the SL operators you've come to know and love (refer to Chapter 4 for more information about these lovable operators).

REMEMBER

SL provides eight *implication rules*, which are rules that allow you to build a safe bridge to get from here to there (premises to conclusion). In other words, you start with one or more true statements and end up with another true one. Most of these rules are simple — they may even seem trivial. But this simplicity is their superpower because it enables you to handle much more complex ideas with great confidence.

To help you remember the implication rules, I break the eight rules into four pairs, as follows:

- **The if-rules:** Modus ponens (**MP**) and modus tollens (**MT**)
- **The and-rules:** Conjunction (**Conj**) and simplification (**Simp**)
- **The or-rules:** Addition (**Add**) and disjunctive syllogism (**DS**)
- **The double-if-rules:** Hypothetical syllogism (**HS**) and constructive dilemma (**CD**)

In the rest of the chapter, I explain these eight rules and show you how to use them to write proofs in SL. Along the way, I share a few tricks for making this task

a bit easier. (See Chapter 12 for more in-depth coverage of proof strategies.) I also provide you with a number of ways to remember these rules when exam time rolls around.

TIP

At the end of the day, you need to memorize these eight implication rules. So work with them for a while, and you may find that you remember most of them without too much effort.

The If-Rules: Modus Ponens and Modus Tollens

In Chapter 3, I compare if-statements to a slippery slide: If the first part of the statement is true, then the second part *must* be true in order for the whole statement to be true. Both implication rules in this section — *modus ponens* (**MP**) and *modus tollens* (**MT**) — take advantage of this idea in different ways.

Modus ponens

MP takes advantage of the slippery slide idea directly. It tells you the following:

> If you know that a statement of the form $x \rightarrow y$ is true, and you also know that the x part is true, then you can conclude that the y part is also true.

In other words, once you step on the top of the slide at x, you have no choice but to end up at y.

REMEMBER

MP: $x \rightarrow y,\ x :\ y$

MP says that *any* argument of this form is valid. Here it is in its simplest form:

$$P \rightarrow Q,\ P :\ Q$$

But, by the same rule, these similar arguments are all valid as well:

$$\sim P \rightarrow \sim Q,\ \sim P :\ \sim Q$$

$$(P \,\&\, Q) \rightarrow (R \,\&\, S),\ P \,\&\, Q :\ R \,\&\, S$$

$$(P \leftrightarrow \sim(Q \,\&\, \sim R)) \rightarrow (\sim S \vee (R \rightarrow P)),\ (P \leftrightarrow \sim(Q \,\&\, \sim R)) :\ (\sim S \vee (R \rightarrow P))$$

REMEMBER

A typical proof has a bunch of numbered rows. The top rows are the premises, the last row is the conclusion, and the rows in between are the intermediate steps that logically connect them. Each row has a line number and a statement, followed by a *justification* for that statement (which identifies the rule and the row numbers it affects).

For example, here's a proof for the argument $(P \& Q) \rightarrow R,\ (P \& Q) :\ R$:

1. $(P \& Q) \rightarrow R$	**P**
2. $(P \& Q)$	**P**
3. R	1,2 **MP**

This proof is simple and didn't require any intermediate steps. The premises $(P \& Q) \rightarrow R$ and $(P \& Q)$ with the justification **P** lead immediately to the conclusion R, with the justification being the rule **MP** applied to lines 1 and 2.

Now try to prove that this slightly more complicated argument is valid:

$$P \rightarrow Q,\ Q \rightarrow R,\ R \rightarrow S,\ P :\ S$$

The first step to any proof is always the same: Copy down all the premises with line numbers and justifications. Here's what your first step should look like:

1. $P \rightarrow Q$	**P**
2. $Q \rightarrow R$	**P**
3. $R \rightarrow S$	**P**
4. P	**P**

After you copy your premises into rows, look for a step that you can take. In this case, you can use **MP** with lines 1 and 4.

5. Q	1,4 **MP**

MP enables you to derive a new statement, Q, which you can use as part of your next step. This time, you can use **MP** with lines 2 and 5.

6. R	2,5 **MP**

Again, you derived a new statement, R, which you can use for your next step. The last step practically writes itself.

7. S	3,6 **MP**

TIP

You know you're finished with the steps in a proof when the conclusion of the argument appears. In this case, S is the conclusion you were looking to justify, so you're done.

Modus tollens

MT uses the slippery slide idea in a different way than **MP** uses it. It tells you the following:

> If you know that a statement of the form $x \rightarrow y$ is true, and you also know that the y part is false, then you can conclude that the x part is also false.

In other words, if you *don't* end up on the bottom of the slide at y, then you *didn't* step onto it at x in the first place.

REMEMBER

MT: $x \rightarrow y,\ \sim y :\ \sim x$

As with **MP** (shown in the previous section), **MT** and all the other rules of inference can be generalized. Thus, a few easily provable valid arguments in this form are as follows:

$$P \rightarrow Q,\ \sim Q :\ \sim P$$
$$(P \,\&\, Q) \rightarrow R,\ \sim R :\ \sim (P \,\&\, Q)$$
$$(P \vee Q) \rightarrow (R \leftrightarrow S),\ \sim (R \leftrightarrow S) :\ \sim (P \vee Q)$$

Knowing this rule, you can now prove the validity of the following argument:

$$P \rightarrow Q,\ \sim P \rightarrow R,\ \sim Q :\ R$$

As always, begin by copying the premises, like this:

1. $P \rightarrow Q$	**P**
2. $\sim P \rightarrow R$	**P**
3. $\sim Q$	**P**

So far, I've given you two rules to work with: **MP** and **MT**. You'll need both in this proof.

TIP

In most proofs, short statements are easier to work with than long ones.

The shortest statement here is $\sim Q$, so a good plan is to hunt around for a way to use it. You don't have to hunt too far, because you can use **MT** as follows:

4. $\sim P$	1,3 **MT**

Now you can use **MP**, like so:

5. R	2,4 **MP**

Again, the proof is finished when the conclusion appears.

The And-Rules: Conjunction and Simplification

The two and-rules are related by a common factor: and-statements. *Simplification* (**Simp**) is useful for breaking down and-statements at the beginning of a proof, and *conjunction* (**Conj**) works for building them up at the end.

Conjunction

Conj says, "If you know two things separately, then you also know both of them together."

REMEMBER

Conj: $x,\ y :\ x \,\&\, y$

Conj is a pretty straightforward rule: If you have two true statements, *x* and *y*, you can conclude that the statement *x* & *y* is also true.

Try this proof on for size:

$$P \rightarrow Q,\ R \rightarrow S,\ P,\ \sim S :\ (P \,\&\, \sim S) \,\&\, (Q \,\&\, \sim R)$$

First, copy the premises, like this:

1. $P \rightarrow Q$	**P**
2. $R \rightarrow S$	**P**
3. P	**P**
4. $\sim S$	**P**

REMEMBER

Writing a proof can be a little like a treasure hunt: Try to find clever ways to find the next clue. For example, study the conclusion so you know where you're trying to go. Then look at the premises and see what will help you get there.

TIP

Generally speaking, long conclusions make for easier proofs than short ones. The strategy is to try to build a long conclusion, one sub-statement at a time.

In this proof, you want to build the sub-statements $(P \,\&\, \sim S)$ and $(Q \,\&\, \sim R)$. The operators in these sub-statements are and-operators, which clues you in that **Conj** can help you build them. In fact, you can get one of them in one step.

5. $P \,\&\, \sim S$	3,4 **Conj**

TIP

Look for statements that have constants in common and see whether you can combine them using the rules of inference.

Looking at the example, you can see that lines 1 and 3 have constants in common, and so do lines 2 and 4.

6. Q	1,3 **MP**
7. $\sim R$	2,4 **MT**

Now you can combine these two statements using **Conj** as follows:

8. $Q \,\&\, \sim R$	6,7 **Conj**

This gives you the other sub-statement of the conclusion. The only thing left to do is to build the conclusion using the pieces you've gathered.

9. $(P \,\&\, \sim S) \,\&\, (Q \,\&\, \sim R)$	5,8 **Conj**

This is a long, nine-line proof, but when you work it piece by piece, it comes together.

TIP

After you walk through this proof as it's presented in this section, I suggest testing your understanding of the process. So, copy down the argument, close the book, and see whether you can do it by yourself. You may find a few sticking points along the way, but it's better to discover them now than during an exam!

Simplification

Simp tells you, "If you have two things together, then you also have either of those things separately."

REMEMBER

Simp: $x \,\&\, y :\ x$
$x \,\&\, y :\ y$

Simp is sort of the flip side of **Conj**. But, instead of starting with the pieces and building to the whole, you start with the whole and reduce to either of the pieces.

Try writing the proof for this argument:

$$P \to Q,\ R \to S,\ P \,\&\, {\sim}S :\ Q \,\&\, {\sim}R$$

As always, copy down your premises, like so:

1. $P \to Q$	**P**
2. $R \to S$	**P**
3. $P \,\&\, {\sim}S$	**P**

TIP

Use **Simp** early in a proof to *unpack* and-statements — that is, turn long and-statements into shorter statements. This gives you easier statements to work with, which you can use to build your conclusion.

Line 3 gives you the only opportunity to use this rule.

4. P	3 **Simp**
5. ${\sim}S$	3 **Simp**

You may notice a certain similarity among the proofs presented in this chapter. If you do, that's good! These next two steps should look pretty familiar by now:

6. Q	1,4 **MP**
7. $\sim R$	2,5 **MT**

All that remains is to put the pieces together, like this:

8. $Q \ \& \sim R$	6,7 **Conj**

The Or-Rules: Addition and Disjunctive Syllogism

As **Simp** is related to **Conj**, *disjunctive syllogism* (**DS**) is related to *addition* (**Add**). Both rules work with or-statements. **DS** breaks them down, and **Add** builds them up. I start with the **Add** rule.

Addition

REMEMBER

Add tells you, "If you know *x*, then you can conclude that either *x* or *y* is the case."

Add: $x : \ x \vee y$

At first glance, this rule can seem downright weird. You may be asking, "If I'm starting only with *x*, where did the *y* come from?" Believe it or not, the beauty of **Add** is the seemingly magical appearance of the *y* out of thin air.

Remember, to build an or-statement that's true, you need only one part of it to be true (see Chapter 4). The other part can be anything you like.

Try out this proof:

$$Q \rightarrow S, \ Q : \ ((P \leftrightarrow \sim Q) \leftrightarrow R) \vee ((P \vee S) \ \& \ (Q \vee R))$$

This proof sure looks odd. You don't have much to work with just two premises.

1. $Q \rightarrow S$	**P**
2. Q	**P**

The next step almost suggests itself.

3. S	1,2 **MP**

Now, to make a start on the difficult-looking conclusion, notice that you can use **Add** to build the following two sub-statements:

4. $P \vee S$	4 **Add**
5. $Q \vee R$	2 **Add**

Next, use **Conj** to put these two statements together:

6. $(P \vee S)\ \&\ (Q \vee R)$	4,5 **Conj**

TIP

When the conclusion of an argument is an or-statement, you only need to build one of the two sub-statements, and then you can use **Add** to tack on the rest.

The key thing to realize for this proof is that you've *already* built the second part of the conclusion: $((P \vee S)\ \&\ (Q \vee R))$. And because the conclusion is an or-statement, you can use **Add** to build the entire conclusion.

7. $((P \leftrightarrow \sim Q) \leftrightarrow R) \vee ((P \vee S)\ \&\ (Q \vee R))$	6 **Add**

If Line 7 seems like a surprising leap, take a moment to think about what's happening in this step: You've already proved that sub-statement $((P \vee S)\ \&\ (Q \vee R))$ is true, so **Add** allows you to prove *any* or-statement built from this sub-statement.

Keep this handy trick in mind — to use **Add** to build a large or-statement from a single sub-statement — especially when the conclusion you're trying to prove is long and convoluted!

Disjunctive syllogism

DS says, "If you have two options and you can eliminate one of them, you can be sure of the one that's left."

REMEMBER

DS: $x \vee y,\ \sim x :\ y$

$x \vee y,\ \sim y :\ x$

DS is related to **Add** in this way: **DS** breaks down or-statements, and **Add** builds them up.

See what you can do with this argument:

$$P \rightarrow {\sim}Q,\ P \vee R,\ Q \vee S,\ {\sim}R :\ {\sim}P \vee S$$

First, copy your premises, like this:

1. $P \rightarrow {\sim}Q$	**P**
2. $P \vee R$	**P**
3. $Q \vee S$	**P**
4. $\sim R$	**P**

In this case, the simplest statement is $\sim R$. Line 2 also contains the constant R. These two statements allow you to use **DS** immediately.

5. P	2,4 **DS**

Now that you have the statement P, the next statement follows easily.

6. $\sim Q$	1,5 **MP**

Then you have another opportunity to use **DS**.

7. S	3,6 **DS**

And, finally, don't miss the opportunity to use **Add**:

8. $\sim P \vee S$	7 **Add**

But wait! Doesn't this proof already *disprove* $\sim P$ in Step 5? Yes, it does. But, as I mention in the previous section, that's the beauty of **Add**: When the conclusion you're trying to prove is an or-statement, as $\sim P \vee S$ is, you can use **Add** to build it from a single sub-statement that you've already proved (in this case, S, which you proved in Step 7).

The Double-If-Rules: Hypothetical Syllogism and Constructive Dilemma

Hypothetical syllogism (**HS**) and *constructive dilemma* (**CD**) allow you to draw conclusions when you start out with two if-statements. You won't use them as often as the other six rules I mention in this chapter, but you'll still need them from time to time.

Hypothetical syllogism

HS tells you, "If you know that *x* leads to *y* and that *y* leads to *z*, then *x* leads to *z*."

REMEMBER

HS: $x \to y,\ y \to z :\ x \to z$

Note that **HS** is the first rule so far that contains no single constants. It neither breaks down statements nor builds them up.

Here's an example for you to try:

$$P \to Q,\ Q \to R,\ R \to S, \sim S :\ \sim P \,\&\, (P \to S)$$

As usual, write out your premises.

1. $P \to Q$	**P**
2. $Q \to R$	**P**
3. $R \to S$	**P**
4. $\sim S$	**P**

Looking at the conclusion, you can see you need to get two pieces — $\sim P$ and $P \to S$ — and then put them together using **Conj**. You can get them in either order. I start by using **HS**.

5. $P \to R$	1,2 **HS**
6. $P \to S$	3,5 **HS**

That gives you the second part of the conclusion. But now, the first part isn't difficult to find.

7. $\sim P$	4,6 **MT**

Then, just put them together using **Conj**, like so:

8. $\sim P \ \& \ (P \rightarrow S)$	6,7 **Conj**

Constructive dilemma

CD is less intuitive than the other rules in this chapter until you really think about it: In plain English, it says, "Suppose you know that you have either *w* or *x*. And you also know that *w* gives you *y* and that *x* gives you *z*. Well, then you have either *y* or *z*."

REMEMBER

CD: $w \vee x,\ w \rightarrow y,\ x \rightarrow z :\ y \vee z$

CD is the only implication rule that uses three statements to produce one, which allows for few opportunities to use it. However, when such an opportunity arises, it's usually flashing at you like neon sign. In other words, it's super easy to spot.

In this example, I give you six premises in a desperate attempt to camouflage this opportunity:

$$P \rightarrow Q,\ Q \rightarrow R,\ S \rightarrow T,\ U \rightarrow V,\ S \vee U,\ \sim R :\ (\sim P \ \& \sim Q) \& (T \vee V)$$

So, first write out your premises.

1. $P \rightarrow Q$	**P**
2. $Q \rightarrow R$	**P**
3. $S \rightarrow T$	**P**
4. $U \rightarrow V$	**P**
5. $S \vee U$	**P**
6. $\sim R$	**P**

At first glance, this example is just a huge mess. One thing to notice, though, is that the conclusion is an and-statement. This means that if you can get both parts of it — $\sim P \ \& \sim Q$ and $T \vee V$ — you can use **Conj** to get the whole thing.

First, use **CD** to get $T \vee V$ as follows:

7. $T \vee V$	3,4,5 **CD**

Here's a play-by-play to make sure you understand this: Statements 3 and 4 give you $S \rightarrow T$ and $U \rightarrow V$. And then statement 5 gives you $S \vee U$, which allows you to apply **CD** and get $T \vee V$.

Now, how do you get $\sim P \,\&\, \sim Q$? Just get $\sim P$ and $\sim Q$ separately, and then use **Conj** to put them together. With this plan in place, begin by proving $\sim Q$, like so:

8. $\sim Q$	2,6 **MT**

This step enables you to prove $\sim P$.

9. $\sim P$	1,8 **MT**

Now, as planned, use **Conj** to put these two sub-statements together.

10. $\sim P \,\&\, \sim Q$	8,9 **Conj**

To complete the proof, use **Conj** again, like so:

11. $(\sim P \,\&\, \sim Q) \;\&\; (T \vee V)$	7,10 **Conj**

As you can see, this 11-line proof, using four of the eight inference rules from this chapter, is relatively advanced. If you're up for the challenge, try writing out this argument, and then close the book and try to prove it. Even if you have to peek a little to complete the task, no worries: If you've understood the steps reasonably well, you're ready to extend your knowledge of SL proofs with the 10 equivalence rules in Chapter 10.

IN THIS CHAPTER

- Understanding a key difference between equivalence rules and implication rules
- Using the ten important equivalence rules

Chapter 10
Applying Equivalence Rules

If you like implication rules (see Chapter 9), you're going to adore the other ten rules of inference that I go over in this chapter: the *equivalence rules*. Why? Let me count the ways.

First of all, these rules will dazzle and astound you (and numerous other admirers) with the ease they provide you in working logic proofs. For example, just try proving the validity of the following argument by using only implication rules:

$$\sim(P \,\&\, Q),\ P:\ \sim Q$$

Alas, you can't. But, luckily for you, when you apply the equivalence rules I explain in this chapter, problems like these are just momentary distractions on an otherwise cloudless day. Even more good news: Equivalence rules are generally easier and more flexible to use within proofs for several important reasons, which I also explain.

So, in this chapter, you discover how to apply ten important equivalence rules, get tips on when and how to use them, and continue to sharpen your skills at proving the validity of arguments. The proofs in this chapter also make full use of the implication rules you know from Chapter 9.

Distinguishing Implications from Equivalences

The implication rules I discuss in Chapter 9 have several key limitations that the equivalence rules in this chapter *don't* have. In this way, the equivalence rules are more flexible and generally more useful than the implication rules. In this section, you find out the differences between these two sets of rules.

Thinking of equivalences as ambidextrous

One of the most important differences between equivalences and implications is how they work: Equivalences work in both directions, whereas implications work in only one direction.

For example, when you know that $x \rightarrow y$ and x are both true, *modus ponens* (**MP**) tells you that y is also true (see Chapter 9). However, reversing that deduction gets you into trouble, because knowing that y is true certainly isn't enough information to decide that $x \rightarrow y$ and x are both true.

Luckily, this limitation doesn't apply to the ten equivalence rules. Equivalence rules tell you that two statements are *interchangeable*: Anywhere you can use one statement, you can use the other.

Applying equivalences to part of the whole

Another difference between equivalences and implications is that you can apply equivalences to part of a statement, but you can't do the same thing with implications. Here's an example of an obvious mistake when you're working with implications:

1. $(P \,\&\, Q) \rightarrow R$	**P**
2. P	1 **Simp** (WRONG!)

Remember that the *simplification* rule (**Simp**) states that $x \,\&\, y : x$. So, the mistake here is thinking that you can apply the implication rule **Simp** to part of line 1 $(P \,\&\, Q)$. Equivalence rules aren't bound by such restrictions, however.

Discovering the Ten Valid Equivalences

I'm sure you're just chomping at the bit to get to know the ten equivalence rules — which you'll need to memorize, by the way. Well, here they are, complete with examples.

TIP

One bit of notation you need to know is the double-colon symbol (**::**). When placed between two statements, this symbol means that the two statements are *equivalent* — that is, you can substitute one for the other when needed.

Double negation

Double negation (**DN**) is simple. It tells you, "If *x* is true, then *not not x* is also true."

DN: $x :: {\sim}{\sim}x$

If you read Chapter 9, you can probably work through the following proof in your sleep, without using any equivalence rules:

$${\sim}P \rightarrow Q, {\sim}Q : P$$

1. $\sim P \rightarrow Q$	**P**
2. $\sim Q$	**P**
3. P	1,2 **MT**

However, every time you negate a negation — for example, when you negate $\sim P$ and change it into P — you're technically required to follow these steps:

1. **Negate ~*P* by changing it into ~~*P*.**
2. **Use DN to change ~~*P* into *P*.**

So, the proof at the beginning of this section is technically missing a step, shown in the following version:

$${\sim}P \rightarrow Q, {\sim}Q : P$$

1. $\sim P \rightarrow Q$	**P**
2. $\sim Q$	**P**
3. $\sim(\sim P)$	1,2 **MT**
4. P	3 **DN**

WARNING

Check whether your teacher is a stickler for this technicality. If so, be careful not to assume a **DN** without putting it explicitly in your proof.

I'm going out on a brittle limb here: Throughout this chapter, I use **DN** frequently without referencing it. My philosophy is that you don't need to get all bent out of shape every time you flip from double negative to positive. So, sue me if you must, but I *don't* know *nothing* about double negation.

Contraposition

Chapter 4 tells you that a statement of the form $x \to y$ and its *contrapositive* $(\sim y \to \sim x)$ always have the same truth value. Chapter 6 clues you in that two statements that always have the same truth value are *semantically equivalent*. These two facts put together give you *contraposition* (**Contra**).

Contra: $x \to y :: \sim y \to \sim x$

Contra is related to *modus tollens* (**MT**), which I introduce in Chapter 9. Each of these rules is saying: "When you start out knowing that the statement $x \to y$ is true, then the fact that $\sim y$ is true leads quickly to $\sim x$."

TIP

An easy way to think of **Contra** is *reverse* and *negate both*. That is, you can *reverse* the two parts of an if-statement as long as you *negate both* parts of it. For example, take a look at this proof:

$$P \to Q,\ \sim P \to R :: \sim R \to Q$$

1. $P \to Q$	**P**
2. $\sim P \to R$	**P**

You have two opportunities to use **Contra**.

3. $\sim Q \to \sim P$	1 **Contra**
4. $\sim R \to P$	2 **Contra**

Now you can complete the proof in one step using *hypothetical syllogism* (**HS**) from Chapter 9.

5. $\sim R \to Q$	1,4 **HS**

By the way, notice that in this proof you didn't use line 3; in fact, you can cross it out now if you like. (Your professor probably won't take off points for showing unnecessary steps, but if you're unsure, then cross it out, then neatly renumber the lines that follow it.)

TIP

You don't have to use every statement or even every premise in your proof (although most of the time you do need them all). But, writing down all the statements you can derive and then seeing which ones you need can be useful. You find out more about this strategy of jotting down the easy stuff in Chapter 12.

Implication

The rationale behind *implication* (**Impl**) is simple: When you know that the statement $x \to y$ is true, you know that either x is false (that is, $\sim x$ is true) or that y is true. In other words, you know that the statement $\sim x \vee y$ is true.

Impl: $x \to y :: \sim x \vee y$

What's also true is that (as with all valid equivalences) this rule works in reverse. So, whenever you're given $x \vee y$, you can change it to $\sim x \to y$. Professors who are sticklers may insist that you first change $x \vee y$ to $\sim\sim x \vee y$ using **DN** and then change this to $\sim x \to y$ using **Impl**.

An easy way to think of **Impl** is *negate the first and change*. For example, you can change an if-statement to an or-statement (or vice versa), as long as you *negate* the *first part* of the statement. Check out this proof:

$P \to Q,\ P \vee R :\ Q \vee R$

1. $P \to Q$	**P**
2. $P \vee R$	**P**

This is a tough little proof because you don't have much to go on and it doesn't have any single-constant statements.

REMEMBER

Impl links every or-statement with an if-statement. Also, **Contra** gives you two versions of every if-statement. That's three forms to work with.

For example, $P \to Q$ has two equivalent forms: $\sim P \vee Q$ by **Impl** and $\sim Q \to \sim P$ by **Contra**. Writing both out involves no cost or obligation, and seeing them explicitly

written out may suggest a path forward in your proof, so be sure to write out all your steps as you see them. Here are the two equivalent forms of $P \to Q$:

3. $\sim P \vee Q$	1 **Impl**
4. $\sim Q \to \sim P$	1 **Contra**

And here are the two equivalent forms of $P \vee R$:

5. $\sim P \to R$	2 **Impl**
6. $\sim R \to P$	2 **Contra**

Now look at what you're trying to prove: $Q \vee R$. With **Impl**, this is the same as $\sim Q \to R$. Aha! So, here are the next steps:

7. $\sim Q \to R$	4,5 **HS**
8. $Q \vee R$	7 **Impl**

You didn't need lines 3 and 6, but they're free, so who cares? (If the answer to this rhetorical question is, "My professor cares — they take off points for extra steps," then go back and cross out lines as needed.)

TIP

When in doubt when writing a proof, write down as many statements as you can. It can't hurt and often helps. I call this strategy *kitchen-sinking* because you're throwing everything but the kitchen sink at the problem. I discuss it and other proof strategies in greater detail in Chapter 12.

REMEMBER

You can apply any equivalence rule to a sub-statement. Here's an example of a proof that exploits this option:

$$\sim P \vee (Q \to R) :\ P \to (\sim Q \vee R)$$

1. $\sim P \vee (Q \to R)$	**P**

With only one premise, you don't have much to work with. Begin by applying **Impl** to this premise, creating a double negative that you can formally simplify using **DN**.

2. $\sim\sim P \to (Q \to R)$	1 **Impl**
3. $P \to (Q \to R)$	2 **DN**

Now, apply **Impl** to the sub-statement $Q \rightarrow R$, changing it to $\sim Q \vee R$:

4. $P \rightarrow (\sim Q \vee R)$	3 **Impl**

TIP

Be sure to look for opportunities to apply equivalence rules to sub-statements.

Exportation

Exportation (**Exp**) isn't intuitively obvious until you put it into words, and then its meaning jumps right out at you.

Exp: $x \rightarrow (y \rightarrow z) :: (x \,\&\, y) \rightarrow z$

To understand **Exp**, think of an example in which $x \rightarrow (y \rightarrow z)$ is in the form of an English statement. Here's one possibility:

> *If* I go to work today, *then if* I see my boss, *then* I'll ask for a raise.

Now think of a similar example with $(x \,\&\, y) \rightarrow z$, such as

> *If* I go to work today *and* I see my boss, *then* I'll ask for a raise.

These two statements essentially mean the same thing, which should tell you that the two statement forms they're based on are semantically equivalent.

WARNING

Don't mix up the parentheses! **Exp** tells you nothing about the statement $(x \rightarrow y) \rightarrow z$ or the statement $x \,\&\, (y \rightarrow z)$.

Try this proof on for size:

$$(P \,\&\, Q) \rightarrow R,\ \sim R \vee S,\ P :\ Q \rightarrow S$$

1. $(P \,\&\, Q) \rightarrow R$	**P**
2. $\sim R \vee S$	**P**
3. P	**P**

Statement 1 is in one of the two forms that **Exp** works with, so write it out.

4. $P \rightarrow (Q \rightarrow R)$	1 **Exp**

Now another avenue opens up because P is one of your premises.

5. $Q \rightarrow R$	3,4 **MP**

Okay, so you have $Q \rightarrow R$, but the conclusion you're trying to build is $Q \rightarrow S$. If you can get your hot little hands on $R \rightarrow S$, you can build the bridge you need using **HS**. Consider these steps:

6. $R \rightarrow S$	2 **Impl**
7. $Q \rightarrow S$	5,6 **HS**

Although this proof has only seven lines, it's a tough one — and it would've been a lot more difficult without **Exp**.

Commutation

You may remember the *commutative property* from arithmetic as the rule telling you that order doesn't affect addition and multiplication — for example, $2 + 3 = 3 + 2$ and $5 \times 7 = 7 \times 5$.

In sentential logic (SL), *commutation* (**Comm**) tells you that order doesn't affect operations with either the and-operator or the or-operator. Thus, **Comm** has two versions.

Comm: $x \,\&\, y :: y \,\&\, x$
$x \vee y :: y \vee x$

Like **DN**, **Comm** may seem so obvious that you think it almost doesn't need to be mentioned. But, unlike **DN**, I think it's worth mentioning when you use it in a proof, and your professor probably will, too.

Here's an example of a proof where **Comm** comes in handy:

$$P \,\&\, (\sim Q \rightarrow R) : (R \vee Q) \,\&\, P$$

1. $P \,\&\, (\sim Q \rightarrow R)$	**P**
2. P	1 **Simp**
3. $\sim Q \rightarrow R$	1 **Simp**
4. $Q \vee R$	3 **Impl**

Here comes your **Comm** opportunity — don't miss it!

5. $R \vee Q$	4 **Comm**
6. $(R \vee Q) \& P$	5 **Conj**

As you can see, using **Comm** together with **Impl** allows you to turn $\sim Q \rightarrow R$ not only into $Q \vee R$, but also into $R \vee Q$. This step allows you to finish up the proof with *conjunction* (**Conj**), which I cover in Chapter 9.

Association

Association (**Assoc**) tells you that for statements with all and-operators or with all or-operators, you can move the parentheses around freely.

Assoc: $(x \& y) \& z :: x \& (y \& z)$
$(x \vee y) \vee z :: x \vee (y \vee z)$

Like **Comm**, **Assoc** has counterparts in arithmetic — for example, (3 + 4) + 5 = 3 + (4 + 5).

TIP

Assoc and **Comm** can be powerful tools when used together. With just these two rules, you can rearrange, in any way you like, any statement that consists of all and-statements or all or-statements. Just be careful that you do it step-by-step.

Here's a proof with just one not very helpful looking premise to try out. As always, write the premise first.

$(P \rightarrow Q) \vee R : Q \vee (R \vee \sim P)$

1. $(P \rightarrow Q) \vee R$	**P**

REMEMBER

Don't forget that you can apply equivalence to sub-statements within a larger statement. In this case, apply **Impl** to the sub-statement $P \rightarrow Q$ in your premise as follows:

2. $(\sim P \vee Q) \vee R$	1 **Impl**

Notice that this result has only or-statements, as does the conclusion. So, as I mention in the Tip in this section, it's possible to rearrange the constants in line 2 to make this statement look like the conclusion using only **Comm** and **Assoc**. Start by using **Comm** to get Q in its proper position as the first constant in the statement.

3. $(Q \vee \sim P) \vee R$	2 **Comm**

Notice that, again, I applied the equivalence rule to part of the statement. Next, I want to move the parentheses to the right using **Assoc**.

4. $Q \vee (\sim P \vee R)$	3 **Assoc**

All that's left to do now is switch around $\sim P$ and R.

5. $Q \vee (R \vee \sim P)$	4 **Comm**

This particular proof also works in reverse — that is, you can use the conclusion to prove the premise. This is true of *any* proof that has only one premise and uses only equivalence rules. It also tells you that the premise and conclusion are semantically equivalent. (See Chapter 8 for more information about semantic equivalence.)

Distribution

As with **Exp**, *distribution* (**Dist**) is confusing until you put it into words, and then it starts to make sense.

Dist: $x \,\&\, (y \vee z) :: (x \,\&\, y) \vee (x \,\&\, z)$
$x \vee (y \,\&\, z) :: (x \vee y) \,\&\, (x \vee z)$

This rule also has a similar form in arithmetic. For example:

$$2 \times (3 + 5) = (2 \times 3) + (2 \times 5)$$

For this reason, multiplication *distributes* over addition. Note that the reverse isn't true.

$$2 + (3 \times 5) \neq (2 + 3) \times (2 + 5)$$

In SL, however, the and-operator and the or-operator distribute over each other. Here's how the and-operator distributes over the or-operator, starting with a statement in English that matches the form $x \,\&\, (y \vee z)$:

> I have a pet *and* it's *either* a cat *or* a dog.

And here's the parallel statement for $(x \,\&\, y) \vee (x \,\&\, z)$:

> *Either* I have a pet *and* it's a cat *or* I have a pet *and* it's a dog.

These two statements mean the same thing, which should help you understand why **Dist** works.

Similarly, here's how the or-operator distributes over the and-operator. This time, I start with a statement that matches the form $x \vee (y \,\&\, z)$.

> I have to take *either* organic chemistry *or both* botany *and* zoology.

The parallel statement for $(x \vee y) \,\&\, (x \vee z)$ is a little awkward to translate, but here it is:

> I have to take *either* organic chemistry *or* botany, *and* I also have to take *either* organic chemistry *or* zoology.

Spend a moment comparing this statement with the previous version to convince yourself that they mean the same thing.

Here's a proof to get you started with **Dist**:

$$Q \vee R,\ \sim(P \,\&\, Q),\ P \therefore R$$

1. $Q \vee R$	**P**
2. $\sim(P \,\&\, Q)$	**P**
3. P	**P**

You can apply **Impl** and **Contra** to line 1, which wouldn't hurt, and I usually do recommend the tactic of writing down every valid statement that occurs to you while you're hunting around for ideas. However, I take this proof in another direction to show you how to use **Dist**.

Start by simply combining lines 1 and 3 using **Conj**.

4. $P \,\&\, (Q \vee R)$	1,3 **Conj**

From here, you can see how this statement is tailor-made for **Dist**.

5. $(P \,\&\, Q) \vee (P \,\&\, R)$	4 **Dist**

Now the wheels are turning. What next?

TIP

When you get stuck in the middle of a proof, look at the premises you haven't used to get ideas about how to move forward.

Notice that the second premise is just the negation of the first part of the or-statement in line 5, so you can use *disjunctive syllogism* (**DS**) from Chapter 9 for your next step.

6. $P \,\&\, R$	2,5 **DS**

Finish up by applying **Simp** to this statement to get the conclusion.

7. R	6 **Simp**

This is a tough little proof, and you may be wondering what you would've done if you hadn't known to use **Conj** to derive the statement in line 4. In the next section, I show you how to tackle this same proof in an entirely different way.

De Morgan's theorem

Like **Dist** and **Exp**, *De Morgan's theorem* (**DeM**) becomes clearer with an example using statements in English.

DeM: $\sim(x \,\&\, y) :: \sim x \vee \sim y$
$\sim(x \vee y) :: \sim x \,\&\, \sim y$

Here's a statement in English that matches the form $\sim(x \,\&\, y)$. Notice that this statement is the *not . . . both* situation I discuss in Chapter 4:

> It's *not true* that I'm *both* rich *and* famous.

And here's the parallel statement matching the form $\sim x \vee \sim y$:

> *Either* I'm *not* rich *or* I'm *not* famous.

As you can see, these two statements mean the same thing, which gives you an intuitive grasp on why the corresponding forms are equivalent.

Here's a statement in English that matches the form $\sim(x \vee y)$. Notice that this statement is the *neither . . . nor* situation I discuss in Chapter 4:

> Jack is *neither* a doctor *nor* a lawyer.

And here's the parallel statement matching $\sim x \,\&\, \sim y$:

> Jack *isn't* a doctor *and* he *isn't* a lawyer.

TIP

In practice, you often use **DeM** to unpack statements of the forms $\sim(x \,\&\, y)$ and $\sim(x \vee y)$ into forms that are easier to work with because they don't have parentheses.

Here's the argument from the preceding section, which you can now prove in a different way by using **DeM**:

$$Q \vee R, \sim(P \,\&\, Q), P \therefore R$$

1. $Q \vee R$	**P**
2. $\sim(P \,\&\, Q)$	**P**
3. P	**P**

This time, you can start by applying **DeM** to the statement in line 2.

4. $\sim P \vee \sim Q$	2 **DeM**

Rewriting this statement without parentheses makes it much easier to work with, so you're more likely to see the next step using **DS**.

5. $\sim Q$	3,4 **DS**

The proof practically finishes itself with another application of **DS**.

6. R	1,5 **DS**

TIP

With your ever-expanding arsenal of equivalence rules, you can almost always find more than one way to get to where you're going within a proof. If your first attempt doesn't work, try another route.

Tautology

Tautology (**Taut**) is the simplest rule in the chapter.

Taut: $x \,\&\, x :: x$

$x \vee x :: x$

In fact, an "Uh-duh!" may be in order here. So why bother?

To be honest, you don't really need **Taut** in the and-statement case. You can change $x \,\&\, x$ to x by using **Simp**, and you can change x to $x \,\&\, x$ by using **Conj**. Similarly, in the or-statement case, you can change x to $x \vee x$ by using **Add**. However, if you're stuck with $x \vee x$ and need to get x, **Taut** is the way to go.

Even so, I can think of only one way you'd be likely to run into this situation, and it's kind of cute. Consider the following statement:

$$P \rightarrow {\sim}P : {\sim}P$$

I can hear you thinking, *Is this proof even possible?* Yes, it is. Take a look:

1. $P \rightarrow {\sim}P$	**P**
2. ${\sim}P \vee {\sim}P$	1 **Impl**
3. ${\sim}P$	2 **Taut**

So, given that the statement $P \rightarrow {\sim}P$ is true, you can prove that ${\sim}P$ is also true. But to do that, you need **Taut**.

TECHNICAL STUFF

By the way, notice that this proof — $P \rightarrow {\sim}P : {\sim}P$ — uses only equivalence rules plus a single premise, so you can reverse it to prove the following argument:

$${\sim}P : P \rightarrow {\sim}P$$

If you're game, cover up the three-step proof you just completed and see if you can prove $\sim P : P \rightarrow \sim P$.

Equivalence

You may think that I lost track of if-and-only-if–statements. If only! The fact is, of the 18 rules of inference I discuss in this chapter and Chapter 9, *equivalence* (**Equiv**) is the only one that includes an if-and-only-if–statement.

Equiv: $x \leftrightarrow y :: (x \rightarrow y) \,\&\, (y \rightarrow x)$
$x \leftrightarrow y :: (x \,\&\, y) \vee (\sim x \;\&\; \sim y)$

Now the truth can be told: I've been ignoring these statements because they're so darned ornery to work with in proofs.

One reason for the relative orneriness of if-and-only-if–statements is that their truth tables are perfectly symmetrical. By contrast to the tables for the other three binary operators (and-, or-, and if-operators), the table for the if-and-only-if–operator divides evenly into two true statements and two false ones. This symmetry is visually pretty, but it isn't much help when you want to narrow a field down to *one* possibility.

So, when an argument contains a premise with an if-and-only-if–operator, you're going to have to unpack it, and the sooner, the better. But you're in luck, because the two **Equiv** rules can help you do just that.

- The first of the two **Equiv** forms exploits the idea that in an if-and-only-if–statement, the arrow goes in both directions. So, it just pulls the statement apart into two if-statements and connects them with an and-operator.
- The second form of **Equiv** exploits the idea that both parts of an if-and-only-if–statement have the same truth value, which means that either *x* and *y* are both true or they're both false.

Take a look at this argument:

$$P \leftrightarrow (Q \,\&\, R),\ Q :\ P \vee \sim R$$

1. $P \leftrightarrow (Q \,\&\, R)$	**P**
2. Q	**P**

TIP

When you have an if-and-only-if–statement as a premise, begin by automatically writing both **Equiv** forms in your proof.

3. $(P \rightarrow (Q \& R)) \& ((Q \& R) \rightarrow P)$	1 **Equiv**
4. $(P \& (Q \& R)) \vee (\sim P \& \sim (Q \& R))$	1 **Equiv**

While you're at it, use **Simp** to unpack the and-statement version of **Equiv** in line 3.

5. $P \rightarrow (Q \& R)$	3 **Simp**
6. $(Q \& R) \rightarrow P$	3 **Simp**

Voilà! You've written four statements automatically. At this point, look over what you have. In this case, line 6 looks like an **Exp** waiting to happen.

7. $Q \rightarrow (R \rightarrow P)$	6 **Exp**

From here, notice that you haven't even touched line 2 yet, and it all falls into place, like so:

8. $R \rightarrow P$	2,7 **MP**
9. $\sim R \vee P$	8 **Impl**
10. $P \vee \sim R$	9 **Comm**

In this case, you didn't need the second **Equiv** form. But check out this argument, where you do need it to complete the proof:

$$P \leftrightarrow Q,\ R \rightarrow (P \vee Q) :\ R \rightarrow (P \& Q)$$

What follows is the most advanced proof in this chapter.

TIP

After reading through this proof, study all the steps to make sure you understand them, and then try to reproduce the proof with the book closed. If you conquer this proof, you'll have a good grasp on some of the most difficult of the 18 rules of inference.

As always, the premises come first.

1. $P \leftrightarrow Q$	**P**
2. $R \rightarrow (P \vee Q)$	**P**

And here are the four statements you can write out without much thought, all unpacked from the first premise:

3. $(P \to Q) \,\&\, (Q \to P)$	1 **Equiv**
4. $(P \,\&\, Q) \vee (\sim P \,\&\, \sim Q)$	1 **Equiv**
5. $P \to Q$	3 **Simp**
6. $Q \to P$	3 **Simp**

Now, notice that line 2 is $R \to (P \vee Q)$, and the conclusion is $R \to (P \,\&\, Q)$. So, thinking strategically, if you can find a way to derive $(P \vee Q) \to (P \,\&\, Q)$, then the implication rule **HS** would give you the conclusion.

The insight here is that line 4 provides a path by allowing you to change an or-statement to an if-statement with **Impl**.

7. $\sim(P \,\&\, Q) \to (\sim P \,\&\, \sim Q)$	4 **Impl**

Here's another difficult step that you may not figure out easily:

8. $\sim(\sim P \,\&\, \sim Q) \to (P \,\&\, Q)$	7 **Contra**

This step gets the sub-statement $(P \,\&\, Q)$ to the second part of the if-statement, bringing you one step closer to proving $(P \vee Q) \to (P \,\&\, Q)$.

Now, clean up the sub-statement $\sim(\sim P \,\&\, \sim Q)$ using **DeM**.

9. $(P \vee Q) \to (P \,\&\, Q)$	8 **DeM**

Okay, the strategy worked! Complete the proof as follows:

10. $R \to (P \,\&\, Q)$	2,9 **HS**

Speaking personally, I'd assign a difficult proof like this as a group project, but probably not as homework, and I certainly wouldn't put it on a test. So, I hope your professor is equally merciful.

If not, hang tough! In Chapters 11 and 12, you discover a variety of strategies and tactics for moving forward with difficult proofs.

IN THIS CHAPTER

- Grasping the basics of conditional proof
- Assuming and discharging premises
- Validating arguments with indirect proof
- Mixing conditional and indirect proof

Chapter 11

Making Big Assumptions with Conditional and Indirect Proof

Have you ever seen one of those cheesy infomercials where the announcer keeps asking, "*Now* how much would you pay?" as they throw in a bunch of extra stuff you don't want with your $19.95 oven-safe goggles — the cheese grater, the ice crusher, and the potato peeler that glows in the dark. This chapter is a bit like those commercials, but with one key difference: I throw in all sorts of extra stuff that you really *do* want.

In this chapter, I introduce two new methods of proof — *conditional proof* and *indirect proof* — completely free of charge. Unlike *direct proof,* which I discuss in Chapters 9 and 10, conditional proof and indirect proof involve an *assumption*, which is an additional premise you don't know is true but you assume to be true. Also in this chapter, at no further cost to you, I show you not only how but also when to use conditional and indirect proof methods.

Conditioning Your Premises with Conditional Proof

So, you've been a diligent student, and you've practiced writing proofs until you can do it in your sleep. You've even memorized every implication and equivalence rule from Chapters 9 and 10. Congratulations — you've made it through the tough stuff alive!

But just when you think you're all set to be your school's resident logician — Wham! — this argument rocks your world:

$$P \rightarrow \sim Q,\ P \vee \sim R : Q \rightarrow ((\sim P\ \&\ \sim R) \vee (Q \rightarrow P))$$

Well, at least you know how to start (by listing the premises, of course).

1. $P \rightarrow \sim Q$	**P**
2. $P \vee \sim R$	**P**

After writing out these premises, you have no doubt that this is going to be one hairy proof. But, what's the harm in giving it the old college try? Here are a few of the statements you come up with along the way:

3. $Q \rightarrow \sim P$	1 **Contra**
4. $\sim Q \vee \sim P$	3 **Impl**
5. $\sim (Q \& P)$	4 **DeM**
6. $\sim (\sim P \& R)$	2 **DeM**
7. $\sim P \rightarrow \sim R$	2 **Impl**
8. $R \rightarrow P$	7 **Contra**
9. $R \rightarrow \sim Q$	1,8 **HS**
10. $Q \rightarrow R$	9 **Contra**
11. $\sim Q \vee R$	10 **Impl**

None of these statements really get you to the conclusion you need to reach. If the path is there at all, it's very well hidden. So, it's time for a new tactic — one that's easy to use and actually works. Yup, you guessed it: The easy-to-use tactic I'm hinting at is conditional proof.

REMEMBER

The conditional proof method often makes a proof easier, but you can't always use it. However, when the conclusion to an argument is an if-statement, conditional proof is usually the way to go.

Understanding conditional proof

Conditional proof (**CP**) allows you to use part of the conclusion as a premise that you can rely on to prove the rest of the conclusion.

REMEMBER

To prove the validity of an argument whose conclusion is in the form $x \rightarrow y$ (that is, any if-statement), you can follow these steps:

1. **Detach the sub-statement *x*.**
2. **Add *x* to your list of premises as an *assumed premise* (AP).**
3. **Prove the sub-statement *y* as if it were the conclusion.**

The idea is simple, but brilliant. So, have another look at the example from the previous section:

$$P \rightarrow \sim Q,\ P \vee \sim R : Q \rightarrow ((\sim P\ \&\ \sim R) \vee (Q \rightarrow P))$$

This time, instead of working your way through and trying to prove that the conclusion $Q \rightarrow ((\sim P\ \&\ \sim R) \vee (Q \rightarrow P))$ is valid, conditional proof allows you to do the following:

- Detach the sub-statement Q.
- Add Q to the list of premises as an **AP**.
- Prove the sub-statement $(\sim P\ \&\ \sim R) \vee (Q \rightarrow P)$ as if it were the conclusion.

REMEMBER

The premises are the same, but now you have an additional *assumed premise* (**AP**). Here's how it works when you list the premises:

1. $P \rightarrow \sim Q$	**P**
2. $P \vee \sim R$	**P**
3. Q	**AP**

With this proof, you're trying to build the statement $(\sim P \,\&\, \sim R) \vee (Q \rightarrow P)$. This time, however, you have a way to break your premises down and get the pieces you need. For example, check out these steps:

4. $\sim P$	1,3 **MT**
5. $\sim R$	2,4 **DS**

With all three single-constant statements at your fingertips, you can go to work.

6. $\sim P \,\&\, \sim R$	4,5 **Conj**

Now, recall from Chapter 9 that you can use **Add** to build any or-statement, provided that you've proved one of its two sub-statements:

7. $(\sim P \,\&\, \sim R) \vee (Q \rightarrow P)$	6 **Add**

To complete the proof, here's the final formality:

8. $Q \rightarrow ((\sim P \,\&\, \sim R) \vee (Q \rightarrow P))$	3–7 **CP**

REMEMBER

This final step is called *discharging the AP*, which makes clear to someone reviewing your proof that, even though you were operating *as if* the assumption were true from statements 3 through 7, you're no longer making this assumption in statement 8. In other words, the conclusion is true even if the assumption *isn't* true!

You may be protesting, "But I can't do that, can I? I didn't prove that the *actual* conclusion is true. All I proved is that *part* of it is true. And worse yet, I did it using a phony premise I stole from the conclusion."

This setup seems too good to be true. After all, premises are like money in the bank, and the conclusion is like a nasty credit card debt you'd rather not look at. But, what if I told you that you can cut your debt in half *and* put money in the bank? That's exactly what conditional proof allows you to do — it's fair, legal, and no credit agency will contact you.

For example, recall that the original argument looked like this:

$$P \rightarrow \sim Q,\ P \vee \sim R :\ Q \rightarrow ((\sim P \,\&\, \sim R) \vee (Q \rightarrow P))$$

If the conclusion could talk, it would say, "You need to show me that if Q is true, then $(\sim P \,\&\, \sim R) \vee (Q \rightarrow P)$ is also true."

Using a conditional proof, you can talk back to the conclusion, saying, "Okay then, I'll show you that assuming Q is true, $(\sim P \ \& \sim R) \vee (Q \rightarrow P)$ is also true." And then you do exactly that: You *assume* that Q is true and you prove the rest.

Tweaking the conclusion

TIP

You can apply equivalence rules to the conclusion of an argument to make conditional proof easier to use.

Flipping conclusions with Contra

To take full advantage of conditional proof, you have to remember that you can use every if-statement in two ways: the way it is originally and the way it is in its *contraposition* (**Contra**) form (see Chapter 10). So, when the conclusion is an if-statement, you can use conditional proof to attack it in two different ways.

For example, check out this proof, which takes the if-statement at face value:

$$P \rightarrow Q,\ R \vee (Q \rightarrow P) :\ \sim (P \leftrightarrow Q) \rightarrow R$$

1. $P \rightarrow Q$	**P**
2. $R \vee (Q \rightarrow P)$	**P**
3. $\sim (P \leftrightarrow Q)$	**AP**

If you begin the proof in this way, your assumed premise isn't very helpful.

But imagine using **Contra** on the conclusion to get the equivalent form $\sim R \rightarrow (P \leftrightarrow Q)$. This allows you to take the following steps:

1. $P \rightarrow Q$	**P**
2. $R \vee (Q \rightarrow P)$	**P**
3. $\sim R$	**AP**

Then you're trying to prove $(P \leftrightarrow Q)$. This solution is much more straightforward.

4. $Q \rightarrow P$	2,3 **DS**
5. $(P \rightarrow Q) \& (Q \rightarrow P)$	1,4 **Conj**
6. $P \leftrightarrow Q$	5 **Equiv**

Now you can discharge your **AP** as follows:

7. $\sim R \rightarrow (P \leftrightarrow Q)$	3–6 **CP**

And don't forget to take the proof all the way to the conclusion, like so:

8. $\sim (P \leftrightarrow Q) \rightarrow R$	7 **Contra**

REMEMBER

Any changes you make to the conclusion will appear at the end of the proof, after you have discharged the assumption.

In this case, even though you thought of using **Contra** first, and you wrote the proof with this in mind, it actually appears last.

Winning through Impl

TIP

You can also turn any or-statement into an if-statement by using *implication* (**Impl**), which I cover in Chapter 10. Thus, using **Impl** makes any or-statement a potential candidate for conditional proof.

For example, consider this argument:

$$P : \sim R \vee (Q \rightarrow (P \,\&\, R))$$

Without conditional proof, you don't have much hope. But the problem becomes much simpler after you notice that you can use **Impl** to rewrite the conclusion as

$$R \rightarrow (Q \rightarrow (P \,\&\, R))$$

And even better, you can use *exportation* (**Exp**) from Chapter 10 to rewrite it again:

$$(R \,\&\, Q) \rightarrow (P \,\&\, R)$$

So, now you're ready to start your proof.

1. P	**P**
2. $R \,\&\, Q$	**AP**

The steps to get $P \,\&\, R$ practically write themselves.

3. R	2 **Simp**
4. $P \,\&\, R$	1,3 **Conj**

With almost no effort, you're ready to discharge your **AP**.

5. $(R \& Q) \rightarrow (P \& R)$	2–4 **CP**

REMEMBER

After you discharge the **AP**, the rest is just tracing back to the original form of the conclusion.

6. $R \rightarrow (Q \rightarrow (P \& R))$	5 **Exp**
7. $\sim R \vee (Q \rightarrow (P \& R))$	6 **Impl**

Stacking assumptions

REMEMBER

After you assume a premise, if the new conclusion is an if-statement (or can be turned into one), you can assume another premise. This is nice way to get two (or more!) assumptions for the price of one. Here's an example:

$$\sim Q \vee R : (P \vee R) \rightarrow ((Q \& S) \rightarrow (R \& S))$$

Start out, as usual, with your premise and **AP**.

1. $\sim Q \vee R$	**P**
2. $P \vee R$	**AP**

Unfortunately, you still have a long way to go to prove $(Q \& S) \rightarrow (R \& S)$. But, because the new conclusion is an if-statement, you can pull out another **AP**, like this:

3. $Q \& S$	**AP**

Now, the new goal becomes proving $R \& S$. Here's what the steps look like:

4. Q	3 **Simp**
5. S	3 **Simp**
6. R	1,4 **DS**
7. $R \& S$	5,6 **Conj**

At this point, you can discharge the last **AP** that you assumed.

8. $(Q \& S) \rightarrow (R \& S)$	3–7 **CP**

And now you can discharge the first **AP**.

9. $(P \vee R) \rightarrow ((Q \& S) \rightarrow (R \& S))$	2–8 **CP**

REMEMBER

When you assume more than one premise in your proof, you must discharge them in reverse order: Discharge the *last* premise *first* and then work your way back to the first premise.

Thinking Indirectly: Verifying Arguments with Indirect Proof

REMEMBER

Indirect proof (**IP**) is an industrial-strength method that works for every proof. However, even though it's sometimes the *only* method that will work, it isn't necessarily always the easiest way to prove an argument, so use it sparingly.

For example, just when you thought everything was going your way, you come across an argument you simply can't figure out:

$$P \rightarrow (Q \ \& \sim R),\ R : \sim (P \ \vee \sim R)$$

1. $P \rightarrow (Q \ \& \sim R)$	**P**
2. R	**P**

Seemingly not much you can do with this argument, huh? And because the conclusion is in a form that you cannot turn into an if-statement, conditional proof is out of the picture, so you may think you're really stuck. The good news is, you're not stuck at all. In fact, a whole new world is about to open up to you — and a beautiful world at that!

This section shows you how to use indirect proof, which unlike conditional proof, is *always* an option no matter what the conclusion looks like.

Introducing indirect proof

Indirect proof (also called *proof by contradiction*) is a type of logical judo. The idea is to assume that the conclusion is false and then show why this assumption is wrong. Success means your conclusion was true all along.

REMEMBER

To show that any argument is valid with indirect proof, you can follow these steps:

1. **Negate the conclusion.**
2. **Add this negation to your list of premises as an assumption.**
3. **Prove any *contradictory statement* (any statement of the form x & ~ x).**

When working on the example from the previous section, instead of trying to prove the conclusion, $\sim(P \vee \sim R)$, use indirect proof, which allows you to rely on its negation, $P \vee \sim R$, as an **AP**. Just remember that now you want to prove this assumption leads to a contradictory statement.

Here's what the proof looks like:

$$P \rightarrow (Q \ \& \sim R),\ R :\ \sim(P \vee \sim R)$$

1. $P \rightarrow (Q \ \& \sim R)$	**P**
2. R	**P**
3. $P \vee \sim R$	**AP**

REMEMBER

Your goal is to prove a statement and its negation, and then use them to build a contradictory and-statement.

With the **AP** at your disposal, the following options suddenly open up:

4. P	2,3 **DS**
5. $Q \ \& \sim R$	1,4 **MP**
6. $\sim R$	**Simp**

At this point, you've derived both R and $\sim R$, so you can build them into a single contradictory statement as follows:

7. $R \ \& \sim R$	2,6 **Conj**

REMEMBER

The assumption has led to an impossible situation, so you know that the **AP** must be false. If $P \vee {\sim}R$ is false, then ${\sim}(P \vee {\sim}R)$ must be true.

8. ${\sim}(P \vee {\sim}R)$	3–7 **IP**

As with conditional proof, you need to discharge the **AP**, which makes clear that, even though you were operating as if the assumption were true from statements 3 through 7, you're no longer making this assumption in statement 8. In fact, the very thing you've proved is that the assumption *isn't* true!

Proving short conclusions

As I mention in Chapter 9, arguments in which the conclusion is shorter than the premises tend to be more difficult to prove than those in which the conclusion is longer, because breaking down long premises can be tricky.

However, indirect proof works especially well when the conclusion is shorter than the premises because the negated conclusion becomes a nice, short premise for you to use. For example, consider this argument:

$${\sim}(({\sim}P \vee Q) \,\&\, R) \rightarrow S,\ P \vee {\sim}R,\ {\sim}Q \vee S \therefore S$$

1. ${\sim}(({\sim}P \vee Q) \,\&\, R) \rightarrow S$	**P**
2. $P \vee {\sim}R$	**P**
3. ${\sim}Q \vee S$	**P**

One way or another, you're going to have to break down the first premise, but it will be easier with some help.

4. ${\sim}S$	**AP**

Immediately, things open up and you can take the following steps:

5. $({\sim}P \vee Q) \,\&\, R$	1,4 **MT**
6. ${\sim}P \vee Q$	5 **Simp**
7. R	5 **Simp**

Remember, you're trying to derive two contradictory statements. But now, the opportunities are more plentiful.

8. P	2,7 **DS**
9. Q	6,8 **DS**
10. S	3,9 **DS**

WARNING

When you're using indirect proof, don't be fooled into thinking you're done after you prove the conclusion. Remember, because you assumed the negation of the conclusion, your goal is to build a contradictory statement.

In this case, the **AP** leads to its own negation, which allows you to complete the proof.

11. S & $\sim S$	4,10 **Conj**
12. S	4–11 **IP**

Superficially, line 12 looks like line 10, but now you've discharged the **AP**, so the proof is complete.

Combining Conditional and Indirect Proof

Students often ask, "If I'm already using an **AP** for a conditional proof, do I have to start over if I want to add an **AP** for an indirect proof?"

The good news is, you don't have to start over, as this example explains:

$$\sim P \,\&\, Q \rightarrow (\sim R \,\&\, S), Q \therefore R \rightarrow P$$

1. $\sim P \,\&\, Q \rightarrow (\sim R \,\&\, S)$	**P**
2. Q	**P**

On your first pass over the proof, you can only manage to get this:

3. $\sim(\sim P \,\&\, Q) \vee (\sim R \,\&\, S)$	1 **Impl**

Maybe the next step is there, but you're not seeing it. So, you have to move on. And because conditional proof is an option, you try it first.

4. R	**AP** (for conditional proof)

Now you're trying to prove P, but you're still not sure how to do this directly. So, you can move on to indirect proof by negating what you're trying to prove and adding it as an assumed premise.

5. $\sim P$	**AP** (for indirect proof)

Then the goal is to find a contradiction. Suddenly, the pieces fall into place.

6. $\sim P \,\&\, Q$	2,5 **Conj**
7. $\sim R \,\&\, S$	1,6 **MP**
8. $\sim R$	7 **Simp**

You've proved both R and $\sim R$, so you're ready to discharge the **AP** for the indirect proof.

9. $R \,\&\, \sim R$	4,8 **Conj**
10. P	5–9 **IP**

Of course, proving P was the goal of the conditional proof, so here's what you get:

11. $R \rightarrow P$	4–10 **CP**

REMEMBER

When using conditional and indirect proof methods together, discharge your **AP**s starting with the last one you added and work your way back to the first.

IN THIS CHAPTER

- Handling easy proofs quickly
- Using conditional proof to work through moderate proofs
- Giving difficult proofs a run for their money

Chapter 12
Polishing Off Your Proofs

Some logic problems almost solve themselves. Others look tough at first but soon fall in line when you know what to do. Still others resist your efforts every step of the way until you finally negotiate a solution. This chapter is all about what the Serenity Prayer calls *the wisdom to know the difference*. And the wisdom you gain will be the calm confidence you need to write sentential logic (SL) proofs with ease when possible and with perseverance when necessary.

Just as the American justice system declares a defendant innocent until proven guilty, I recommend the easy-until-proven-difficult approach with proofs. In this chapter, I show you how to make a quick assessment of an argument to get a gut sense of how tough it will be to prove. Next, I describe a few quick-and-dirty moves that allow you to handle the easy proofs in five minutes or less.

For more stubborn proofs, I explain how and when to pull out the *conditional proof* (**CP**) technique from Chapter 11. **CP** is often enough to complete proofs of moderate difficulty.

And for the really tough proofs, I show you how to use the form of an SL statement to your advantage by breaking down long premises and working both ends against the middle to complete the proof. I also guide you through an advanced method of indirect proof.

Easy Proofs: Taking a Gut Approach

You don't need to grab a baseball bat to swat a mosquito. Similarly, you don't need to devise a brilliant proof strategy to solve an easy problem. You just need to spend five minutes looking at what's in front of you and jotting down a few ideas. The following sections give you a few quick tricks for writing simple proofs with grace and speed.

Looking at the problem

Looking really does mean just looking. All of the three suggestions I discuss in the following sections should take less than a minute to complete, but it's a quality minute.

Compare the premises and conclusion

Do the premises look somewhat similar to the conclusion, or are they very different? For example, do the premises and conclusion both contain only and-statements or or-statements? Or do a variety of operators, including if-and-only-if–operators, appear in the statements? If the premises and conclusion look similar, the proof may not be so difficult; otherwise, it may be trickier.

In either case, think about what has to happen to bridge the gap between the premises and conclusion. Do you have any hunches about how to proceed?

That's all this step requires — thinking and working from your gut.

Notice the length of the premises and conclusion

REMEMBER

Generally speaking, short premises and a long conclusion indicate an easy proof, whereas long premises and a short conclusion indicate a more difficult proof.

When given enough short premises, you can build almost any conclusion you like. However, on the flip side, breaking down long premises enough to get a short conclusion can be tricky.

In the upcoming section "Jotting down the easy stuff," I show you a bunch of ways to break down premises. For now, just get a gut sense of how tough the proof may be based on this rule of thumb: *The shorter the premises, the easier the proof.*

Look for repeated chunks of statements

TIP

When you notice chunks of statements that are repeated in an argument, underline them so they stand out. The best strategy for these chunks is often to leave them alone rather than break them down.

For example, look at this argument:

$$(P \leftrightarrow Q) \rightarrow \sim(R \,\&\, S),\ R \,\&\, S :\ \sim(P \leftrightarrow Q)$$

You can hammer away at the statement $(P \leftrightarrow Q) \rightarrow \sim(R \,\&\, S)$ to break it down, but there's no reason to. After you notice that the chunks $(P \leftrightarrow Q)$ and $(R \,\&\, S)$ are repeated in the statement, this one-step solution presents itself:

1. $(P \leftrightarrow Q) \rightarrow \sim(R \,\&\, S)$	**P**
2. $R \,\&\, S$	**P**
3. $\sim(P \leftrightarrow Q)$	1,2 **MT**

For now, just noticing these sorts of chunks will help you speed your way through proofs. See Chapter 9 for more information about applying *modus tollens* (**MT**).

Jotting down the easy stuff

With one minute down (as outlined in the previous sections), this section gives you four more one-minute strategies to move quickly through easy proofs. I call this type of strategy *kitchen-sinking*, because you're using everything in your proof except the kitchen sink.

TIP

As you use these strategies, don't hesitate to go for the finish if you see a clear way to complete your proof.

Break down statements using Simp and DS

REMEMBER

Simplification (**Simp**) and *disjunctive syllogism* (**DS**) are the two simplest rules for breaking down and-statements and or-statements (see Chapter 9 for all your **Simp** and **DS** needs). The more you break down premises early on, the better your chances for building up the conclusion.

Expand your options with Impl and Contra

Use *implication* (**Impl**) from Chapter 10 to convert or-statements to if-statements, and vice versa. Then use *contraposition* (**Contra**), also explained in Chapter 10, to convert every if-statement to its contrapositive (which I introduce in Chapter 4). Rely on these equivalence rules to rewrite every statement you can because they give you easy ways to expand your options in either direction for later.

Use MP and MT wherever possible

Opportunities to use the implication rules *modus ponens* (**MP**) and **MT** (as seen in Chapter 9) are easy to spot. These rules tend to give you simple statements to work with.

Convert all negative statements with DeM

REMEMBER

Generally speaking, *De Morgan's theorem* (**DeM**; see Chapter 10) is the only rule that allows you to convert the four negative forms of SL statements to positive ones.

DeM works directly on statements of the forms $\sim(x \,\&\, y)$ and $\sim(x \vee y)$, as I explain in Chapter 10. But, even when you're up against if-statements and if-and-only-if–statements, the two remaining negative forms, you can use **DeM** after employing a few other rules to get these statements into the forms $\sim(x \,\&\, y)$ and $\sim(x \vee y)$.

For example, to convert $\sim(x \rightarrow y)$, use these steps:

1. $\sim(x \rightarrow y)$	**P**
2. $\sim(\sim x \vee y)$	1 **Impl**
3. $x \,\&\, \sim y$	2 **DeM**
4. x	3 **Simp**
5. $\sim y$	3 **Simp**

In just a few steps, you've turned a complex-looking statement into two simple ones.

Even when you're saddled with the dreaded form $\sim(x \leftrightarrow y)$, you can use *equivalence* (**Equiv**), which I cover in Chapter 10, and then break every piece down with **DeM**. For example, take a look at these steps:

1. $\sim(x \leftrightarrow y)$	**P**
2. $\sim((x \,\&\, y) \vee (\sim x \,\&\, \sim y))$	1 **Equiv**
3. $\sim(x \,\&\, y) \,\&\, \sim(\sim x \,\&\, \sim y)$	2 **DeM**
4. $\sim(x \,\&\, y)$	3 **Simp**
5. $\sim(\sim x \,\&\, \sim y)$	3 **Simp**

At this point, you can use **DeM** again to further simplify statements 4 and 5. Sure, it's a few extra steps, but you've turned an impenetrable statement into two very simple ones.

Knowing when to move on

Suppose you've spent about five minutes with a particularly hairy problem. You've looked at it and turned it over in your head once or twice. You've already jotted down a few simple statements — or maybe you didn't see any to jot down — and now you're just about out of ideas.

My advice: five minutes. That's all I'd give the gut strategy. And that goes double if the premises are long and the conclusion is short. If the proof doesn't jump out at you in five minutes, you need a better game plan so the next five minutes are productive rather than frustrating.

TIP

Even if you decide to move on to a new tactic, you don't have to start over. Any statement you've already proved is yours to keep and use throughout the rest of the proof.

Moderate Proofs: Recognizing When to Use Conditional Proof

The better game plan I promised in the previous section is coming right up. After you've abandoned hope that the problem is simple, it's time to pull out the conditional proof method (which I introduce in Chapter 11).

TIP

Try conditional proof first (after the gut approach doesn't do the trick) whenever possible because it tends to be the quickest way to a solution for a problem of medium difficulty.

To decide when conditional proof is possible, look at the conclusion you're trying to prove and decide which of the eight basic statement forms it demonstrates. (Check out Chapter 5 for a rundown on all eight basic forms of SL statements.)

You can always use conditional proof for three of the eight basic forms, which I call the *friendly forms*. You can also use conditional proof for two other forms — the *slightly-less-friendly forms* — but more work is necessary. And finally, you can't use conditional proof for the remaining three forms, which I refer to as the *unfriendly forms*.

Friendly Forms	Slightly-Less-Friendly Forms	Unfriendly Forms
$x \to y$	$(x \leftrightarrow y)$	$x \,\&\, y$
$x \vee y$	$\sim(x \leftrightarrow y)$	$\sim(x \to y)$
$\sim(x \,\&\, y)$		$\sim(x \vee y)$

In this section, I discuss the cases in which you can use conditional proof. I save the remaining cases for the section "Difficult Proofs: Knowing What to Do When the Going Gets Tough."

The three friendly forms: $x \to y$, $x \vee y$, and $\sim(x \,\&\, y)$

Obviously, you can always use conditional proof on any conclusion in the form $x \to y$ by taking x as your conditional premise (CP) and proving y. But you can also easily use conditional proof on any conclusion in the forms $x \vee y$ and $\sim(x \,\&\, y)$.

REMEMBER

You can turn any conclusion of the form $x \vee y$ into an if-statement just by using **Impl**. This rule turns $x \vee y$ into $\sim x \to y$.

For example, suppose you want to prove this argument:

$$R : \sim(P \,\&\, Q) \vee (Q \,\&\, R)$$

1. R	**P**

With only one short premise, you don't have much to go on here. But, after you realize that the conclusion is equivalent to $(P \& Q) \rightarrow (Q \& R)$, you can use conditional proof, like this:

2. $P \& Q$	**AP**
3. Q	2 **Simp**
4. $Q \& R$	1,3 **Conj**
5. $(P \& Q) \rightarrow (Q \& R)$	2–4 **CP**

After your *assumed premise* (**AP**) is discharged, all that's left to do is show how the statement you just built is equivalent to the conclusion you're trying to prove. (Flip to Chapter 11 for guidance on working with **AP**s.)

6. $\sim(P \& Q) \vee (Q \& R)$	5 **Impl**

REMEMBER

The other friendly form that works with conditional proof is $\sim(x \& y)$. When your conclusion is in this form, you need to use **DeM** to get it out of the negative form, changing it to $\sim x \vee \sim y$. From there, you can use **Impl** to change it to $x \rightarrow \sim y$.

For example, suppose you want to prove this argument:

$$\sim P \rightarrow Q, P \rightarrow \sim R : \sim(\sim Q \& R)$$

1. $\sim P \rightarrow Q$	**P**
2. $P \rightarrow \sim R$	**P**

TIP

The insight here is to realize that the conclusion is equivalent to $Q \vee \sim R$ (by **DeM**), which is then equivalent to $\sim Q \rightarrow \sim R$ (by **Impl**). Again, you can use conditional proof, like this:

3. $\sim Q$	**AP**
4. P	1,3 **MT**
5. $\sim R$	2,4 **MP**
6. $\sim Q \rightarrow \sim R$	3–5 **CP**

As with previous examples, after you've discharged your **AP**, you need to complete the bridge from the statement you built to the conclusion you started with.

7. $Q \vee \sim R$	6 **Impl**
8. $\sim(\sim Q \& R)$	7 **DeM**

The two slightly-less-friendly forms: $x \leftrightarrow y$ and $\sim(x \leftrightarrow y)$

If I've said it once, I've said it a thousand times: Working with if-and-only-if–statements is always a little dicey. But, luckily, the principle here is the same as with the friendly forms.

TIP

Always remember (and please don't ever forget) that the first step with an if-and-only-if–statement is to get rid of the if-and-only-if–operator by using **Equiv**. Knowing this step will get you halfway home if you happen to get lost.

To work with a conclusion that's in the form of $x \leftrightarrow y$, you first have to use **Equiv** to change it to $(x \& y) \vee (\sim x \ \& \sim y)$. You should recognize the resulting or-statement as a friendly form, which allows you to use **Impl** to change it to $\sim(x \& y) \rightarrow (\sim x \ \& \sim y)$.

For example, suppose you want to prove this argument:

$$((\sim P \vee Q) \vee \sim R) \rightarrow \sim(P \vee R) \therefore P \leftrightarrow R$$

1. $((\sim P \vee Q) \vee \sim R) \rightarrow \sim(P \vee R)$	**P**

I'm not going to kid you: This is a tough proof. Without finding a way to turn the conclusion into a conditional form, it's just about hopeless. Fortunately, you can use **Equiv** to change the conclusion to $(P \& R) \vee (\sim P \ \& \sim R)$, and from there, make it $\sim(P \& R) \rightarrow (\sim P \ \& \sim R)$ using **Impl**. Check it out:

2. $\sim(P \& R)$	**AP**
3. $\sim P \vee \sim R$	2 **DeM**

Now you're trying to prove $\sim P \ \& \sim R$. The big question at this point is, how do you use that hairy premise in line 1? (This is why I say that long premises make for difficult proofs.)

Okay, first things first: You can at least unpack the premise a little bit using **DeM** on the second part.

4. $((\sim P \vee Q) \vee \sim R) \rightarrow (\sim P \ \& \sim R)$	1 **DeM**

The second part of this statement looks just like what you're trying to prove for your conditional proof. So, if you can build the first part of the statement, you can use **MP** to get the second part. The goal, then, is to prove the statement $(\sim P \vee Q) \vee \sim R$.

But now you're probably wondering how you get a Q from nothing at all. Not to worry — you can use *addition* (**Add**) from Chapter 9 to tack a Q onto $\sim P \vee \sim R$.

5. $(\sim P \vee \sim R) \vee Q$	3 **Add**

The next part of this proof just requires a bunch of manipulation using *association* (**Assoc**) and *commutation* (**Comm**), which I explain in Chapter 10.

6. $\sim P \vee (\sim R \vee Q)$	5 **Assoc**
7. $\sim P \vee (Q \vee \sim R)$	6 **Comm**
8. $(\sim P \vee Q) \vee \sim R$	7 **Assoc**

Finally, you get a glimpse of the Holy Grail: You can use line 8 together with line 4 to derive what you need to discharge your assumed premise, like so:

9. $\sim P \ \& \sim R$	4,8 **MP**
10. $\sim (P \& R) \rightarrow (\sim P \ \& \sim R)$	2–9 **CP**

As usual, after discharging the **AP**, you need to make the statement you just built look like the conclusion.

11. $(P \& R) \vee (\sim P \ \& \sim R)$	10 **Impl**
12. $P \leftrightarrow R$	11 **Equiv**

Okay, that was a bear of a proof. But turning the tough conclusion into a more manageable form made it doable.

Proving conclusions that are in the slightly-less-friendly forms can be challenging. The example in this section is certainly pushing the envelope of what I'd call

medium difficulty. I walk you through an example of how to prove a conclusion in the form $\sim(x \leftrightarrow y)$ later in this chapter, in the "Difficult Proofs: Knowing What to Do When the Going Gets Tough" section.

The three unfriendly forms: *x & y, ~ (x → y),* and *~ (x ∨ y)*

REMEMBER

When the conclusion falls into the unfriendly-form category, conditional proof is almost never an option because, as a rule, you can't turn these three forms into the form $x \rightarrow y$.

To understand why you can't use conditional proof in these cases, first notice that no rule exists for turning a statement that's in the form $x \,\&\, y$ into an if-statement. Similarly, statements in the two remaining unfriendly forms are easy to turn into and-statements, but again, you get stuck if you try to turn them into if-statements.

TIP

When you're faced with a conclusion that's in an unfriendly form, my advice is to move on and attempt direct or indirect proof. In either case, you're probably looking at a difficult proof.

Difficult Proofs: Figuring Out What to Do When the Going Gets Tough

When a gut attempt at direct proof fails, and you can't use conditional proof, you're probably facing a difficult proof. In these cases, you're faced with the choice of trying to push forward with direct proof or switching over to indirect proof.

The upcoming sections show you some strategies that help you get through the really tough proofs.

Choosing carefully between direct and indirect proof

TIP

If you come across a problem whose conclusion can't be turned into an if-statement (which would allow you to use conditional proof), consider direct proof your preferred option *except* when you hit one of the three exceptions I list in this section.

Even if you don't find a direct proof in the first five minutes, stick with it for a while longer when you can't use conditional proof. In the best-case scenario, using some of the ideas later in this chapter, you may find direct proof without switching to indirect proof. And at worst, if you need to switch to indirect proof, you can still use all the statements you came up with while looking for direct proof.

WARNING

This switcheroo (between using direct and indirect proof) doesn't work in the reverse direction: If you switch to indirect proof too early and then want to abandon ship later and try direct proof again, you can't use any of the statements you found while looking for indirect proof.

Exception #1: When the conclusion is short

Indirect proof can be very helpful when the conclusion is a short statement and most or all of the premises are long. In this case, turning a troublesome short conclusion into a helpful short premise with indirect proof is a win-win situation.

Exception #2: When the conclusion is long but negated

Indirect proof is especially good for handling long conclusions that are in negative forms — $\sim(x \vee y)$ and $\sim(x \to y)$ — because a negated conclusion will become a positive premise. For example:

$$P,\ Q :\ \sim((Q \vee R) \to (\sim P \,\&\, R))$$

1. P	**P**
2. Q	**P**
3. $(Q \vee R) \to (\sim P \,\&\, R)$	**AP**

Using indirect proof changes the conclusion to a positive form and then adds it as an assumed premise. Now you can begin to chip away at it.

4. $Q \vee R$	2 **Add**
5. $\sim P \,\&\, R$	3,4 **MP**
6. $\sim P$	5 **Simp**
7. $P \,\&\, \sim P$	1,6 **Conj**
8. $\sim((Q \vee R) \to (\sim P \,\&\, R))$	3–8 **IP**

Exception #3: When all hope seems lost

The third place to use indirect proof is when you've banged away at the problem with direct proof for a while and you're not getting anywhere. Just wait a while is all I ask. Converting from direct to indirect proof is always easy. And even if you're using conditional proof already, you can always convert to indirect proof just by adding on another **AP**. (See Chapter 11 for an example of how **AP**s work.)

Working backward from the conclusion

I've compared writing a proof to building a bridge that gets you from here to there (from premises to conclusion). And usually, that's exactly what happens: You're able to get from here to there. But, sometimes, it doesn't work out as well. So, if you find yourself in a bind and you can't get there from here, it may be easier to get from *there* to *here.*

In Chapter 11, in "Flipping conclusions with Contra," I show you a quick example of working backward from the conclusion. In this section, you use this skill to write a very difficult proof. For example, suppose you want to prove this argument:

$$P \to Q,\ (P \to R) \to S,\ (\sim Q \vee \sim S)\ \&\ (R \vee \sim S) :\ \sim(Q \leftrightarrow R)$$

The conclusion is of the form $\sim(x \leftrightarrow y)$, one of the two slightly-less-friendly forms. So, if you want to use conditional proof, you're going to have to rearrange the conclusion. In this case, writing down the *end* of the proof before the beginning is a good idea.

99. $\sim(Q \leftrightarrow R)$

You begin by numbering the last line of the proof *99*. You're not going to need that many lines, but you can renumber them when you're finished. As you can see, the last line contains the conclusion in all its glory.

Then start thinking backward about how you may have ended up at that conclusion in particular. In this case, the best way to do the proof is to use conditional proof, which means the conclusion has to be an if-statement. And the first step would be to use **Equiv** to turn the statement into the friendly form $\sim(x\ \&\ y)$.

98. $\sim((Q \to R)\ \&\ (R \to Q))$	
99. $\sim(Q \leftrightarrow R)$	98 **Equiv**

Now, you have to decide what step came before the one you just figured out. This time, you use **DeM** to get statement 97 out of its negative form.

97. $\sim(Q \to R) \vee \sim(R \to Q)$	
98. $\sim((Q \to R) \,\&\, (R \to Q))$	97 **DeM**
99. $\sim(Q \leftrightarrow R)$	98 **Equiv**

After you get the hang of it, you can see that the next step is to turn statement 97 into an if-statement by using **Impl**.

95. $\sim(R \to Q)$	
96. $(Q \to R) \to \sim(R \to Q)$	4–95 **CP**
97. $\sim(Q \to R) \vee \sim(R \to Q)$	96 **Impl**
98. $\sim((Q \to R) \,\&\, (R \to Q))$	97 **DeM**
99. $\sim(Q \leftrightarrow R)$	98 **Equiv**

After all that, you have the Holy Grail in your hot little hands. And you know how the story ends: Having assumed $Q \to R$ and used it to build $\sim(R \to Q)$, you discharge this assumption by joining these two sub-statements using the if-operator.

And now, of course, you know a lot more about how the story begins. In particular, you know that your **AP** should be $Q \to R$. So, you know your beginning steps would look like this:

1. $P \to Q$	**P**
2. $(P \to R) \to S$	**P**
3. $(\sim Q \vee \sim S) \,\&\, (R \vee \sim S)$	**P**
4. $Q \to R$	**AP**

Now the goal is building the statement $\sim(R \to Q)$, which is the last backward step you figured out. When you look at your **AP** and line 1, *hypothetical syllogism* (**HS**) from Chapter 9 probably jumps out at you:

5. $P \to R$	1,4 **HS**

Then you look at line 2, and you feel like it's your birthday.

6. S	2,5 **MP**

What now? Line 3 is the only premise left, and the form probably looks very, very familiar. So, you break out *distribution* (**Dist**), which you know from Chapter 10.

7. $(\sim Q \,\&\, R) \vee \sim S$	3 **Dist**

Then you get another break.

8. $\sim Q \,\&\, R$	6,7 **DS**

When you hit this point in the proof, it pays to know how to manipulate the eight basic forms.

9. $\sim(Q \vee \sim R)$	8 **DeM**
10. $\sim(\sim R \vee Q)$	9 **Comm**
11. $\sim(R \rightarrow Q)$	10 **Impl**

At this point, you've reached your goal, and all you need to do is renumber the last few rows, like so:

12. $(Q \rightarrow R) \rightarrow \sim(R \rightarrow Q)$	4–11 **CP**
13. $\sim(Q \rightarrow R) \vee \sim(R \rightarrow Q)$	12 **Impl**
14. $\sim((Q \rightarrow R) \,\&\, (R \rightarrow Q))$	13 **DeM**
15. $\sim(Q \leftrightarrow R)$	14 **Equiv**

Digging deeper into SL statements

By now, you're probably getting good at noticing which of the eight forms a statement falls into. Sometimes, though, this isn't quite enough. When a proof is difficult, solving it often depends on understanding the structure of statements at a deeper level.

You may have noticed that three of the equivalence rules break a statement into three parts (*x*, *y*, and *z*) rather than just two. These rules, which I introduce in

Chapter 10, are *exportation* (**Exp**), *association* (**Assoc**), and *distribution* (**Dist**). You probably haven't used these as much as some of the other rules, but you definitely need them for the tough proofs. After you start looking for opportunities to use **Exp**, **Assoc**, and **Dist**, you'll find them all around you.

For example, look at these three statements:

$(P \vee Q) \vee R$

$(P \& Q) \vee R$

$(P \& Q) \vee (R \vee S)$

Can you spot that all these statements have the same basic form $x \vee y$? But don't let this similarity in basic structure blind you to important differences in the deeper structure of the statements.

For example, you can apply **Assoc** to the first statement but not the second, and **Dist** to the second but not the first. And you can apply both **Assoc** and **Dist** to the third statement. Knowing these differences and applying the correct rule becomes essential when proofs get difficult. Read the following sections to get some tips on noticing and exploiting these differences.

Using Exp

Exp: $x \to (y \to z)$ is equivalent to $(x \& y) \to z$

For example, check out this statement in the form $x \to y$:

$(P \& Q) \to (R \to S)$

But you can also look at it in two other ways. One possibility, taking $(P \& Q)$ as a single chunk, is to notice that the statement looks like $x \to (y \to z)$. In this case, you can use **Exp** to change the statement to

$((P \& Q) \& R) \to S$

A second option, taking $(R \to S)$ as a single chunk, is to notice that the statement looks like $(x \& y) \to z$. This time, you can use **Exp** in the other direction, like this:

$P \to (Q \to (R \to S))$

Combining Assoc with Comm

Assoc: $(x \& y) \& z$ is equivalent to $x \& (y \& z)$

$(x \vee y) \vee z$ is equivalent to $x \vee (y \vee z)$

For example, consider this statement:

$$\sim(P \vee Q) \rightarrow (R \vee S)$$

One way to go here is to apply **Impl** so you get this:

$$(P \vee Q) \vee (R \vee S)$$

Now, you can apply **Assoc** in two different directions.

$$P \vee (Q \vee (R \vee S))$$
$$((P \vee Q) \vee R) \vee S$$

You can then use **Comm** to rearrange the four constants in a bunch of different ways. For example, just working with $P \vee (Q \vee (R \vee S))$, you can get the following statements:

$$P \vee (Q \vee (S \vee R))$$
$$P \vee ((R \vee S) \vee Q)$$
$$(Q \vee (R \vee S)) \vee P$$

TIP

If you can express a statement using only or-operators (or only and-operators), you can use a combination of **Assoc** and **Comm** to rearrange the constants in a statement in any order you like, which can be a very powerful tool that helps you shape a statement into just about anything you need.

Getting the gist of Dist

Dist: $x \,\&\, (y \vee z)$ is equivalent to $(x \,\&\, y) \vee (x \,\&\, z)$

$x \vee (y \,\&\, z)$ is equivalent to $(x \vee y) \,\&\, (x \vee z)$

Dist also has these two forms worth knowing, with the sub-statement in parentheses at the front:

$(x \vee y) \,\&\, z$ is equivalent to $(x \,\&\, z) \vee (y \,\&\, z)$

$(x \,\&\, y) \vee z$ is equivalent to $(x \vee z) \,\&\, (y \vee z)$

REMEMBER

Most professors find these other two forms of the rule acceptable. However, a few sticklers may make you use **Comm** to turn $(x \vee y) \,\&\, z$ into $z \,\&\, (x \vee y)$ before applying **Dist**. Or, similarly, they may require you to use **Comm** to turn $(x \,\&\, y) \vee z$ into $z \vee (x \,\&\, y)$ before applying **Dist**.

For example, suppose you have this statement:

$$P \vee (Q \,\&\, (R \vee S))$$

You can use **Dist** two ways. First, taking $(R \vee S)$ as a single chunk, notice that the statement looks like $x \vee (y \,\&\, z)$. Now you can rewrite the statement so that it looks like this:

$$(P \vee Q) \,\&\, (P \vee (R \vee S))$$

The advantage here is that you can use **Simp** to separate the statement into two smaller statements.

$$P \vee Q$$
$$P \vee (R \vee S)$$

The second option is to use **Dist** on the sub-statement $Q \,\&\, (R \vee S)$, which would give you

$$P \vee ((Q \,\&\, R) \vee (Q \,\&\, S))$$

Now, you have three sub-statements — P, $(Q \,\&\, R)$, and $(Q \,\&\, S)$ — joined by or-statements, which means you can use **Assoc** and **Comm** to arrange them in any order you like (as long as you keep the stuff inside the parentheses intact).

TIP

One great use of **Dist** is that it allows you to change the main operator of a statement from *and* to *or*. By changing the main operator in this way, you can turn an unfriendly conclusion — that is, one on which you can't use conditional proof — into a friendly one.

For example, consider this statement:

$$P \,\&\, (Q \vee R)$$

In most cases, when you're faced with a conclusion in the form $x \,\&\, y$, you can forget about conditional proof. But, in this case, you can use **Dist** to change it to

$$(P \,\&\, Q) \vee (P \,\&\, R)$$

Now you have the conclusion in a friendly form, and you can use **Impl** to turn it into an if-statement.

$$\sim(P \,\&\, Q) \rightarrow (P \,\&\, R)$$

Another unfriendly form is $\sim(x \rightarrow y)$. Knowing this form is unfriendly may cause you to abandon conditional proof on the following conclusion:

$$\sim(P \rightarrow \sim(Q \vee R))$$

But, fortunately, you can get this conclusion out of its negative form in two steps. First, use **Impl** to make it look like this:

$$\sim(\sim P \vee \sim(Q \vee R))$$

Next, use **DeM** to get this statement:

$$P \,\&\, (Q \vee R)$$

Surprisingly, this statement is the same as the one I started out with in the example five statements ago, so you can use **Dist** as you did there to transform it to a friendly form, and then attack it with conditional proof.

Breaking down long premises

As I've mention several times, breaking down long premises can be difficult. Sometimes, though, there's no way around it. For example, take a look at this argument:

$$(P \,\&\, Q) \vee (Q \rightarrow R),\ Q,\ \sim R :\ P$$

1. $(P \,\&\, Q) \vee (Q \rightarrow R)$	**P**
2. Q	**P**
3. $\sim R$	**P**

The key to this one — whether you go with direct or indirect proof — is to find a way to break down that long first premise. I suggest direct proof.

TIP

When working with a long premise, identify which form of SL statement it is. Doing so often helps you decide on the next step you should take.

The form of the first premise is $x \vee y$. When you're faced with an or-statement, first try using **Impl**.

4. $\sim (P \,\&\, Q) \rightarrow (Q \rightarrow R)$	1 **Impl**

Then you can look at this statement as $x \rightarrow (y \rightarrow z)$, which allows you to try **Exp**.

5. $(\sim (P \,\&\, Q) \,\&\, Q) \rightarrow R$	4 **Exp**

You're in good shape now because you've isolated R as the second part of the statement. So, you can use **MT**.

6. $\sim (\sim (P \,\&\, Q) \,\&\, Q)$	3,5 **MT**

WARNING

Be careful that you don't try to cancel out the two not-operators here. The first one applies to the entire statement, whereas the second one applies only to the sub-statement $(P \& Q)$.

Now the form of the statement is $\sim(x \& y)$, which means you've come across a good time to use **DeM**.

7. $(P \& Q) \vee \sim Q$	6 **DeM**

The rest of the steps suggest themselves.

8. $P \& Q$	2,7 **DS**
9. P	8 **Simp**

Note that keeping track of the form of the statement was instrumental in peeling away layer after layer of this long premise. Sometimes, when students see tough proofs like these, they ask, "But what if I just don't see it? And what if I can't figure out the next step?"

REMEMBER

The good news is, you can almost always find more than one way to work a proof. So, even if you and another student start out differently on the same problem, you can usually still find a way to make your proof work.

Here's the new proof of the previous argument. The first three steps, listing the argument's three premises, are the same:

$$(P \& Q) \vee (Q \rightarrow R),\ Q,\ \sim R :\ P$$

1. $(P \& Q) \vee (Q \rightarrow R)$	**P**
2. Q	**P**
3. $\sim R$	**P**

This time, however, apply **Impl** to the second part of the statement instead of to the main operator. (Recall from Chapter 10 that you're allowed to apply equivalence rules like **Impl** to part of a statement.)

4. $(P \& Q) \vee (\sim Q \vee R)$	1 **Impl**

This time, you can look at statement 4 as a form of $x \vee (y \vee z)$, which means you can use **Assoc**.

5. $((P \,\&\, Q) \vee \sim Q) \vee R$	4 **Assoc**

Surprise, surprise — you're again in the position to use **DS**. However, you can use it twice in a row in this proof.

6. $(P \,\&\, Q) \vee \sim Q$	3,5 **DS**
7. $P \,\&\, Q$	2,6 **DS**

And, once again, the answer emerges.

8. P	7 **Simp**

Making a shrewd assumption

In Chapter 11, I show you how to use indirect proof by assuming the negation of the conclusion and then disproving it.

But, with indirect proof, you aren't limited to using the negation of the conclusion. In fact, you can make *any* assumption and try to disprove it. If you're successful, then you've proved the *negation* of the assumption, which can often help you prove the conclusion.

Although you can make any assumption, the strategy is to pick an assumption that will quickly lead to a contradiction. For example, here's the argument I proved valid in the previous section "Breaking down long premises":

$$(P \,\&\, Q) \vee (Q \rightarrow R),\ Q, \sim R :\ P$$

1. $(P \,\&\, Q) \vee (Q \rightarrow R)$	**P**
2. Q	**P**
3. $\sim R$	**P**

In this proof, you're looking for a quick way to break down the first premise. This time, you create an assumed premise out of thin air that will help make this happen.

4. $\sim(P \,\&\, Q)$	**AP**

As with any indirect proof, now you're looking for a contradiction. But this takes only a few lines.

5. $Q \rightarrow R$	1,4 **DS**
6. R	2,5 **MP**
7. $R \,\&\, \sim R$	3,6 **Conj**

As usual, the next step is to discharge your **AP**.

8. $P \,\&\, Q$	4–7 **IP**

Now, completing the proof is almost trivial.

9. P	8 **Simp**

IN THIS CHAPTER

- Recognizing the sufficiency of the SL operators
- Creating SL statements with fewer than five operators
- Appreciating the Sheffer stroke

Chapter 13
Putting Operators in Their Place

In Chapter 4, I explain that the word *or* has two different meanings in English: The inclusive *or* means "either this or that, or both," whereas the exclusive *or* means "either this or that, but not both." I also note that the or-operator in sentential logic (SL) removes this ambiguity because it always represents the inclusive *or*. At the time, you may have thought this was highly unfair and discriminatory. In fact, the more rebellious among you may have considered starting a movement to add a sixth operator to SL.

But, before you take to the streets carrying homemade signs and chanting, "Two, four, six, eight, the exclusive *or* is really great," read this chapter. In these pages, you find out how the exclusive *or* — as well as any other homemade operator you may come up with — is already covered by the five SL operators. In fact, in this chapter, I show you how these five symbols allow you to express *any* possible truth function you care to devise.

You also discover how you can express any possible truth function with *fewer* than five SL operators. In fact, you may be surprised at how much you can do with so little.

Making Do with the Five SL Operators

After you've been working with the five SL operators for a while, you may begin to wonder whether having a few more operators would be helpful. In this section, I show you that the answer is a resounding "No!"

REMEMBER

In order to demonstrate this point, I invent a fictional operator: the exclusive or–operator (?). The exclusive or–operator will work just like an exclusive *or* (see Chapter 4), which means *either . . . or . . . but not both.* Here's a truth table for this new operator:

x	y	$x \ ? \ y$
T	**T**	**F**
T	**F**	**T**
F	**T**	**T**
F	**F**	**F**

REMEMBER

You can use this newfangled operator just like you would use any other operator in SL. For example, you can combine it with one of the old familiar operators to make a statement like $(P?Q) \rightarrow P$.

You can even use a truth table to discover which interpretations make the value of this statement **T** or **F**.

P	Q	$(P$	?	$Q)$	$\rightarrow$	P
T	**T**	**T**	**F**	**T**	**T**	**T**
T	**F**	**T**	**T**	**F**	**T**	**T**
F	**T**	**F**	**T**	**T**	**F**	**F**
F	**F**	**F**	**F**	**F**	**T**	**F**

It seems like a great idea. So why hasn't it caught on? Well, because you don't need it. You can get the same result by using standard SL operators.

x	y	$x ? y$	$\sim$	$(x$	$\leftrightarrow$	$y)$
T	T	F	F	T	T	T
T	F	T	T	T	F	F
F	T	T	T	F	F	T
F	F	F	F	F	T	F

As the preceding table shows in the $x ? y$ and $\sim$ columns, the statements $x ? y$ and $\sim x \leftrightarrow y$ are semantically equivalent (see Chapter 6 for more on semantic equivalence). And when two statements are equivalent, you can substitute one for the other.

For example, you can substitute the statement $\sim(P \leftrightarrow Q) \rightarrow P$ for the statement $(P ? Q) \rightarrow P$. As the following table verifies (see the two $\rightarrow$ columns), these two statements are also semantically equivalent.

P	Q	$(P$	?	$Q)$	$\rightarrow$	P	$\sim$	$(P$	$\leftrightarrow$	$Q)$	$\rightarrow$	P
T	T	T	F	T	T	T	F	T	T	T	T	T
T	F	T	T	F	T	T	T	T	F	F	T	T
F	T	F	T	T	F	F	T	F	F	T	F	F
F	F	F	F	F	T	F	F	F	T	F	T	F

So, you really don't need an exclusive or–operator to represent the exclusive *or*. The five operators are sufficient for all your exclusive *or* needs.

In fact, *any* operator you can possibly invent would be equally unnecessary. That's because the five SL operators can represent any statement you may wish to express in SL without resorting to additional operators.

Downsizing the Operators — a Semi-true Story

In the preceding section, I explain that the exclusive or–operator (?, which I made up!) is unnecessary by showing you how to represent the statement $(x ? y)$ as $\sim(x \leftrightarrow y)$. In other words, I demonstrate that these two statements are semantically equivalent.

TIP

When two statements are semantically equivalent, you can substitute one for another whenever you choose. This swapping power comes in handy when you're doing proofs. It also causes an unexpected result, as the fable in the upcoming sections illustrates.

In this section, you use your knowledge of the equivalence rules (see Chapter 10) to see how you can eliminate one SL operator after another and still express what you need to express in SL.

The tyranny of power

Suppose that you're the boss who oversees the five logical operators. One day, you face the fact that you're getting tired of how the if-and-only-if–operator has been behaving. It's late to work all the time, and it's always taking sick days. You want to say, "You're fired!" but you're worried that the other four operators won't be able to do the job by themselves. Then you get an idea.

Under the *equivalence* (**Equiv**) rule (see Chapter 10), any statement of the form $x \leftrightarrow y$ is equivalent to $(x \,\&\, y) \vee (\sim x \,\&\, \sim y)$. Because of this equivalence, you decide to give the other operators fancy new titles and shift around their responsibilities. From now on, instead of using the if-and-only-if–operator, you can rely on a combination of other variables and operators.

For example, instead of writing

$$P \,\&\, (Q \leftrightarrow R)$$

you write

$$P \,\&\, ((Q \,\&\, R) \vee (\sim Q \vee \sim R))$$

Similarly, instead of writing

$$(P \rightarrow \sim Q) \leftrightarrow (R \,\&\, S)$$

you write

$$((P \rightarrow \sim Q) \,\&\, (R \,\&\, S)) \vee (\sim (P \rightarrow \sim Q) \,\&\, \sim (R \,\&\, S))$$

REMEMBER

Not using the if-and-only-if–operator can make writing equivalent statements a little awkward, but it works. In fact, the great discovery here is that you're now able to do with four operators everything that you used to do with five.

The blow of insurrection

It's lonely at the top — even for logical operators. First thing Monday morning, the if-operator shuffles into your office without knocking. It doesn't like the new arrangement one bit, and after much shouting, it gives you an ultimatum: "Either rehire the if-and-only-if–operator or fire me!"

Of course, you don't take kindly to that type of talk, so you have the if-operator escorted out of the building by a burly pair of parentheses. (This is officially where the story takes on an allegorical life of its own.) After the heat of the moment cools a bit, you realize you have another hole to fill.

But again, with the *implication* (**Impl**) rule (see Chapter 10), you can always replace any statement of the form $x \to y$ with the equivalent statement $\sim x \vee y$. This means, for example, that you can rewrite the statement

$$((P \to Q) \,\&\, R) \to S$$

as

$$\sim((\sim P \vee Q) \,\&\, R) \vee S$$

And the whole system is up and running again. You can do everything with three operators (*not*, *and*, and *or*) that you used to do with five.

TECHNICAL STUFF

These three operators are sufficient to express all SL statements, which is of major importance in Boolean algebra, the first rigorous formalization of logic. You can read more about Boolean algebra in Chapter 14.

The horns of dilemma

Just when things seem back to normal in the operator factory, the and-operator and the or-operator request a meeting. They're working longer hours and they know you need them, so they both want big raises.

You offer the operators cigars and brandy and tell them you'll place their request at the top of the agenda at the next board of directors' meeting. After they're out of your office, you devise a plan to get rid of one of them so you can give the other a 50 percent raise and still come out ahead on the deal.

You realize that using *De Morgan's theorem* (**DeM** from Chapter 10), you can replace any statement of the form $x \,\&\, y$ with the equivalent statement $\sim(\sim x \vee \sim y)$. But, then again, you can also replace any statement of the form $x \vee y$ with the equivalent statement $\sim(\sim x \,\&\, \sim y)$.

For the first time, you hesitate. You even consider firing both of them and rehiring the if-operator (figuring you can replace $x \,\&\, y$ with $\sim(x \rightarrow \sim y)$ and $x \vee y$ with $\sim x \rightarrow y$).

REMEMBER

In any case, you now have a lot of room to negotiate. The following three combinations would allow just two operators to do the job of all five:

~ and &

~ and $\vee$

~ and $\rightarrow$

The (Sheffer) stroke of genius

You didn't see it coming. In walks your most faithful employee, the not-operator, to give one month's notice. And this time, it isn't about the money, the hours, or even the office mood. It wants nothing more, in fact, than an early retirement with a halfway decent settlement package.

With this news, you may have to close up shop for good. Even if all four of the other operators come back, you can't negate a statement without the not-operator.

Just when everything looks most unfortunate, a surprise visitor appears: the *nand-operator* (|). This operator is short for *not and*, and it's sometimes called the *Sheffer stroke*, after its inventor, Henry Sheffer. The statement $x \mid y$ is semantically equivalent to the statement $\sim(x \,\&\, y)$.

x	y	$x \mid y$	$\sim$	$(x$	&	$y)$
T	T	F	F	T	T	T
T	F	T	T	T	F	F
F	T	T	T	F	F	T
F	F	T	T	F	F	F

Hiring on this new help provides some distinct advantages. For example, with this operator, you can write a negative expression, such as $\sim x$, using the expression $x \mid x$.

x	$\sim x$	$x \mid x$
T	F	F
F	T	T

You can also write and-statements, such as $x \ \& \ y$, using the expression $(x \mid y) \mid (x \mid y)$. Additionally, you can write or-statements, $x \vee y$ for example, using the expression $(x \mid x) \mid (y \mid y)$.

REMEMBER

In fact, the Sheffer stroke lets you write every expression the five SL operators allow with just one operator. For example, take the expression

$$P \leftrightarrow Q$$

Start out by expressing the if-and-only-if–operator using a combination of other SL operators:

$$(P \ \& \ Q) \vee (\sim P \ \& \ \sim Q)$$

Next, carefully express each of the two and-operators using only Sheffer strokes:

$$((P \mid Q) \mid (P \mid Q)) \vee ((\sim P \mid \sim Q) \mid (\sim P \mid \sim Q))$$

Now, even more carefully rewrite the or-operator using Sheffer strokes:

$$(((P \mid Q) \mid (P \mid Q)) \mid ((P \mid Q) \mid (P \mid Q))) \mid (((\sim P \mid \sim Q) \mid (\sim P \mid \sim Q)) \mid ((\sim P \mid \sim Q) \mid (\sim P \mid \sim Q)))$$

Finally, use Sheffer strokes to rewrite each of the negative variables, replacing every $\sim P$ with $P \mid P$ and every $\sim Q$ with $Q \mid Q$:

$$(((P \mid Q) \mid (P \mid Q)) \mid ((P \mid Q) \mid (P \mid Q))) \mid (((P \mid P) \mid (Q \mid Q)) \mid ((P \mid P) \mid (Q \mid Q))) \mid (((P \mid P) \mid (Q \mid Q)) \mid ((P \mid P) \mid (Q \mid Q))))$$

All right, it's a little tedious, but it can be done. So, after much thought, you proclaim, "You're hired!" Cue the flashing lights and applause.

The moral of the story

REMEMBER

In reality, the five SL operators are in little danger of being fired, and I highly doubt they'll be retiring anytime soon, which is a darn good thing. Even though the Sheffer stroke (nand-operator) is logically capable of handling all of SL, the final example in the previous section shows you just how eye-crossing these statements would be without the other operators.

Here's an analogy: If you're computer savvy, you may know that everything you do on a computer is translated into 1s and 0s. Your computer keypad, however, contains far more than just these two keys.

This redundancy may be logically unnecessary, but it makes using a computer much easier (just like the five SL operators make reading and writing logical statements so much easier). You can use your natural language rather than the computer's language.

And, in a similar way, the five SL operators closely parallel words such as *and*, *or*, *not*, and so on, to make it easier for you to think about the meaning of what you're doing rather than the rules for doing it.

IN THIS CHAPTER

- Knowing the difference between syntax and semantics
- Discovering well-formed formulas
- Seeing the symbolic connection between Boolean algebra and SL

Chapter 14
Syntactical Maneuvers and Semantic Considerations

George Carlin once remarked that you never hear people say, "Hand me that piano." This statement is funny because, on the grammatical level, it's a perfectly normal sentence and, on the *meaning* level, it's completely absurd. This chapter is all about these two levels:

- *Syntax,* which is the level of grammar, is where Carlin's sentence masquerades as normal. After all, "Hand me that piano." is a grammatical sentence, from the capital letter at the beginning to the period at the end. Syntax is all about the form, or internal structure, of language and the rules that make language possible.
- *Semantics,* which is the level of meaning, is where Carlin's sentence reveals itself to be ridiculous. In other words, you can hand someone a pen, a wallet, or even a skunk, but you just can't hand a person a piano. Semantics is all about the functionality, or practical external usage, of language. In this chapter, it's about the meaning that makes language worthwhile.

Whether you're describing a natural language, such as English, or an invented language, such as sentential logic (SL), syntax and semantics both play vital roles. In this chapter, I clarify the key distinction between syntax and semantics in SL. I discuss the rules for well-formed formulas (WFFs) in SL, which describe how to tell an SL statement from a mere string of symbols. Finally, I introduce Boolean algebra, which is an earlier system of logic.

Working WFF Us or Against Us

I have an easy question for you: Which of the following is a valid statement in SL?

$A)\quad (P \vee Q) \rightarrow \sim R$

$B)\quad \vee R\, Q \rightarrow) \sim (P$

If you chose statement *A*, you're correct. Now, a more difficult question: How did you know?

You may be tempted to respond with a resounding "Uh, duh!" However, it's worth noticing that *A* and *B* contain all the same symbols. They even contain the same number of spaces between the symbols. Okay, *B* looks like *A* after it went through a blender. But is that really a good enough reason to reject it as an SL statement?

Actually, it's a very good reason.

REMEMBER

SL, like any other language, is more than just a bunch of symbols thrown together in any order. The order of its symbols is one of the traits that allow a language to function. And if order wasn't so important, you could write an English sentence like this: Dog food than Empire State Building cobra wine goblet Barbra Streisand glass pagoda fdao udos keowe !voapa-aifaoidao-faid, ; s; j?jj;ag u,R.

Your eyes aren't playing tricks on you — that last "sentence" makes no sense. It should remind you, however, that English also has rules for how you can combine its symbols. These rules are commonly called grammar, and even though you may say you don't have the faintest clue about it, even as you read this book, your brain is crunching through grammatical rules with great speed and accuracy.

Grammar — also known as *syntax* — is simply a set of rules for determining how you can properly order discrete chunks of language. In written language, these chunks are called words and punctuation. In SL, these chunks are called constants, operators, and parentheses.

In this section, you discover how to tell a statement in SL from a clever pretender posing as a statement. You also find out about the precise rules for building a string of SL symbols to evaluate whether a statement is valid.

Understanding WFFs (with a few strings attached)

Any random series of SL symbols is called a *string*. Some strings are statements and others aren't. Deciding whether a string is a statement is a *syntactic* question — that is, a question about the form of the string.

When you're talking about syntax, replacing the word *statement* with the term *well-formed formula* is common. Notice that the word *form* appears twice in this term to hammer home the understanding that the form (in other words, the syntax) of the string is in question. The phrase *well-formed formula* is shortened to *WFF*, which logicians in the know pronounce as "wiff."

REMEMBER

Every string of SL symbols is either a WFF or a non-WFF.

Make no mistake: In SL, the words *statement* and *WFF* mean exactly the same thing. But, when you get to talking about strings at your next cocktail party or logician soiree, you'll want to say *WFF* instead of *statement*, so that's what I do here.

You already know intuitively that $(P \vee Q) \rightarrow \sim R$ is a WFF, but that $\vee RQ \rightarrow) \sim (P$ isn't. And, in this case, your intuition is correct. But what exactly fuels this intuition? Here, I show you three simple rules for turning that intuition into knowledge.

REMEMBER

With these three simple rules, you can build any of the SL statements you see in this book, and any SL statements you'll ever see in the future. And just as important, when you follow these rules, you can't build strings that aren't WFFs. Without further ado, here are the three WFF-ensuring rules:

- **Rule 1:** Any constant (*P, Q, R,* and so on) is a WFF.
- **Rule 2:** If any string x is a WFF, then the string $\sim x$ is also a WFF.
- **Rule 3:** If any two strings x and y are both WFFs, then the strings $(x \,\&\, y)$, $(x \vee y)$, $(x \rightarrow y)$, and $(x \leftrightarrow y)$ are all WFFs.

So, give it a shot: How do you build $(P \vee Q) \rightarrow \sim R$?

From Rule 1, P, Q, and R are all WFFs.

P	Q	R	Rule 1

Then, from Rule 2, the string ~R is also a WFF.

P	Q	$\sim R$	Rule 2

Rule 3 says that the string $(P \vee Q)$ is a WFF.

$(P \vee Q)$	$\sim R$	Rule 3

And, then applying Rule 3 again, the string $((P \vee Q) \rightarrow \sim R)$ is a WFF.

$((P \vee Q) \rightarrow \sim R)$	Rule 3

Relaxing the rules

Technically speaking, every WFF that includes at least one of the four binary operators (*and*, *or*, *if*, and *if-and-only-if*) should begin and end with parentheses. But, in practice, this is rarely the case.

REMEMBER

Removing the outer parentheses from a WFF results in what logicians call the *relaxed* version of that WFF.

As depicted in the example in the previous section, the rules show that the string $((P \vee Q) \rightarrow \sim R)$ is a WFF. Is that WFF the same thing as $(P \vee Q) \rightarrow \sim R$? Strictly speaking, it isn't. So, Rule 4 — also known as the *relaxation rule* — takes care of this.

By convention, with this fourth rule, you can remove the outer parentheses from a WFF to create a relaxed version of that WFF. Technically, though, the relaxed version of a WFF isn't a WFF, and Rule 4 isn't a rule. Instead, it's a convention that makes WFFs a little easier to read.

WARNING

You may *not* use this relaxed version to build new WFFs. So, for example, suppose you have these two WFFs:

$(P \& Q)$	$(Q \vee R)$

You can drop the parentheses, *or* you can build a new WFF by combining these two WFFs:

$P \,\&\, Q$	$Q \vee R$	$((P \,\&\, Q) \rightarrow (Q \vee R))$

You can even build a new WFF and *then* drop the outer parentheses:

$$(P \,\&\, Q) \rightarrow (Q \vee R)$$

But you're *not allowed* to drop the parentheses *first* and *then* build a new WFF:

$(P \,\&\, Q \rightarrow Q \vee R)$	NOT ALLOWED!

As the last statement above demonstrates, the result of dropping the parentheses first and then building a new WFF is an ambiguous statement that can be interpreted in a variety of ways. In short, the rules for building new WFFs are designed to prevent ambiguity, so that every statement can be interpreted in only one way.

Separating WFFs from non-WFFs

Remember that the purpose of the rules for building SL statements is not only to allow you to build WFFs, but also to prevent you from building strings that resemble WFFs (but really aren't). Take a look at the following messy string:

$$\vee RQ \rightarrow)\sim(P$$

You don't have much chance of building this wacky string to create a WFF. But here's another string that looks more like it may be a WFF:

$$(P \vee \sim Q) \rightarrow (R \vee S) \;\&\; \sim T$$

Can you build this statement using the rules? Here's how you can try:

P	Q	R	S	T	Rule 1
P	$\sim Q$	R	S	$\sim T$	Rule 2
$(P \vee \sim Q)$	$(R \vee S)$	$\sim T$			Rule 3

So far, you've accounted for all the parentheses and constants you wanted to add, but you're still missing two operators. And you can't get both operators into the statement without ambiguity — unless you add another set of parentheses.

And so, the string $(P \vee \sim Q) \rightarrow (R \vee S)\ \&\ \sim T$ that you started out to build into a WFF isn't one because its main operator could be either the if-operator or the and-operator. This ambiguity is a definite no-no because — as I cover in Chapter 4 — an SL statement can have only one main operator.

Comparing SL to Boolean Algebra

In Chapter 13, I show you how you can get along in SL with only three operators (*negation*, *and*, and *or*) and still do everything you did with five operators.

The earliest version of formal logic — that is, logic with symbols rather than English words — made use of the fact that you can get along with only three operators. As I note in Chapter 2, Boolean algebra, invented by George Boole, was the first attempt at turning philosophical logic into a rigorous mathematical system. In fact, this form of logic was more closely linked to the mathematics taught in school than SL is.

Believe it or not, Boolean algebra is much easier than the algebra you learned (or didn't) in school. For one thing, you work with only two numbers: 0 and 1. You also only have to worry about addition and multiplication. In this section, I highlight the similarities between SL and Boolean algebra.

Reading the signs

Boolean algebra is really just a version of SL that uses different symbols, so in this section I explore Boolean algebra by making a few changes to SL symbols. In fact, I change only five symbols, which I outline in Table 14-1.

TABLE 14-1 Corresponding Symbols in SL and Boolean Algebra

SL Symbol	Boolean Algebra Symbol
T	**1**
F	**0**
~	–
&	×
∨	+

REMEMBER

Constants are used the same way in Boolean algebra and SL, so when using the new symbols, you can draw the same basic truth tables you're used to (see Chapter 6). For example, you can draw a truth table linking the and-operator in SL with Boolean multiplication.

P	*Q*	*P* & *Q*
T	T	T
T	F	F
F	T	F
F	F	F

P	*Q*	*P* × *Q*
1	1	1
1	0	0
0	1	0
0	0	0

WARNING

Make sure you know that the and-operator in SL is the multiplication symbol (×) in Boolean algebra. Because & means *and*, you may mistakenly associate this operator with addition.

Notice that Boolean multiplication, symbolized by the times sign (×), is exactly the same as regular multiplication with 1s and 0s: $1 \times 1 = 1$ and anything multiplied by 0 is equal to 0.

Similarly, you can draw a truth table linking the or-operator in SL with Boolean addition.

P	*Q*	*P* ∨ *Q*
T	T	T
T	F	T
F	T	T
F	F	F

P	*Q*	*P* + *Q*
1	1	1
1	0	1
0	1	1
0	0	0

REMEMBER

In this case, Boolean addition works like regular addition, but with one exception: In Boolean addition, $1 + 1 = 1$ because 1 is simply the highest number in the system.

Finally, you can draw a truth table linking the not-operator with Boolean negation.

P	~*P*
T	F
F	T

P	–*P*
1	0
0	1

REMEMBER

You may find this strange: In Boolean algebra $-1 = 0$ and $-0 = 1$. But remember that 1 and 0 in Boolean algebra are similar to **T** and **F** in SL. So, obviously, thinking that ~**T** = **F** and ~**F** = **T** isn't so strange.

Using the equals sign (=) with the other mathematical symbols is quite natural. Equality in Boolean algebra means the same thing as truth value in SL. You can also think of the equals sign as meaning that two statements are *semantically equivalent*, which is to say that they have the same truth value no matter what value the variables stand for (see Chapter 6 for more information about semantic equivalence).

Table 14-2 shows the connection between the assignment of truth value in SL and equality in Boolean algebra.

TABLE 14-2 Comparing SL and Boolean Algebra

SL Truth Value or Semantic Equivalence	Boolean Equality
The truth value of P is **T**.	$P = 1$
The truth value of Q is **F**.	$Q = 0$
The truth value of $P \vee \sim P$ is **T**.	$P + -P = 1$
$P \,\&\, Q$ is semantically equivalent to $Q \,\&\, P$.	$P \times Q = Q \times P$
$\sim(P \vee Q)$ is semantically equivalent to $\sim P \,\&\, \sim Q$.	$-(P + Q) = -P \times -Q$

Doing the math

In SL, you avoid writing strings that mix constants with the values **T** and **F**. In Boolean algebra, this avoidance isn't necessary. In fact, you can discover a lot about logic by mixing these different types of symbols.

For example, $P \times 0 = 0$ reminds you that any and-statement — remember, the multiplication sign stands in for the and-operator — that contains a false sub-statement (0) is false, no matter what the rest of the statement looks like.

Similarly, the equation $P + 0 = P$ tells you that when an or-statement contains a false sub-statement, its truth value depends on the value of the other sub-statement. And the equation $P \times 1 = P$ tells you that when an and-statement contains a true sub-statement (1), its truth value also depends on the value of the other sub-statement.

Finally, the equation $P+1=1$ tells you that when an or-statement contains a true sub-statement, the statement is always true.

Running rings around things

REMEMBER

Boolean algebra and arithmetic in the nonnegative integers (0, 1, 2, 3, and so on) are *semirings*, which means they share a set of common properties.

For example, notice that the first three equations in the previous section are also true in regular arithmetic. They work simultaneously on two levels — as expressions of logical truth and as expressions of arithmetic truth.

In Chapter 10, I tell you that in SL, the commutative, associative, and distributive properties from arithmetic carry over into logic. Table 14-3 shows a short list of important properties in Boolean algebra, including these three properties and two others. This set of five properties is sufficient to classify both Boolean algebra and conventional arithmetic as an example of a *semiring*.

TABLE 14-3 Properties Common to Boolean Algebra and Arithmetic (and All Other Semirings)

Property	Addition	Multiplication
Identity element	$P+0=P$	$P\times 1=P$
Annihilator	n/a	$P\times 0=0$
Commutative	$P+Q=Q+P$	$P\times Q=Q\times P$
Associative	$(P+Q)+R=P+(Q+R)$	$(P\times Q)\times R=P\times(Q\times R)$
Distributive	$P\times(Q+R)=(P\times Q)+(P\times R)$	n/a

Exploring syntax and semantics in Boolean algebra

Boolean algebra provides an interesting opportunity to focus on some of the differences between syntax and semantics I discuss in the section "Working WFF Us or Against Us" earlier in this chapter.

REMEMBER

Boolean algebra and SL are quite different with regard to syntax, but very similar in terms of semantics.

On the level of syntax, Boolean algebra and SL are as different as French and German: Even if you understand one of them, you still have to figure out the language syntax if you want to understand the other.

But, after you know the syntactic rules of both, you realize that you can use both systems to express similar sorts of ideas. And looking at how symbols express ideas is the very heart of semantics. Notice how a statement in Boolean algebra has two separate meanings that are independent of each other. For example, consider the following statement:

$$1 \times 0 = 0$$

On one level, this statement expresses an arithmetic truth. In other words, when you multiply 1 by 0, the result is 0. But on another level, it expresses a logical truth, meaning that when you connect a true statement with a false statement using the word *and*, the result is a false statement. Depending on which semantic context you're operating in, an equation in Boolean algebra works equally well to express a mathematical truth and a logical truth.

4 Questing After Quantification Logic

IN THIS PART . . .

Get the basics of quantification logic (QL), including two new quantification operators, ∀ and ∃.

Find out how to translate statements from English into QL.

Rely on the skills you develop with SL to write proofs in QL.

Master two tools for making QL more expressive: relations and identities.

Figure out how to use truth trees in QL, and uncover the surprising truth about infinite trees.

IN THIS CHAPTER

- Getting an overview of quantification logic (QL)
- Going beyond SL with the universal and existential quantifiers
- Distinguishing statements from statement forms

Chapter 15

Expressing Quantity with Quality: Quantification Logic

In Chapter 3, I show you that logic is all about deciding whether an argument is valid or invalid. So, if you've read Chapter 3 and I ask you to take a look at the following argument, you can probably tell me that it's a perfectly valid argument.

Premises:

All of my children are honest.

At least one of my children is a lawyer.

Conclusion:

At least one honest lawyer exists.

This argument is indeed valid. However, the problem arises when you try to use sentential logic (SL) to show that it's valid. None of these three statements contains familiar words such as *and*, *or*, *if*, or *not*, so you can't use the five SL operators to express them.

The best you can do is express the statements as SL constants. For example:

Let *C* = All of my children are honest.

Let *L* = At least one of my children is a lawyer.

Let *H* = At least one honest lawyer exists.

After you express the statements as constants, you can put together the following argument:

C, *L*: *H*

Clearly, something important is lost in translation from English to SL; no matter which method you use — truth table, quick table, truth tree, or proof — this SL argument is invalid. So, either the original argument really is invalid (which it isn't) or SL is wrong (which it isn't).

This chapter (as well as Chapters 16 through 19) gives you a way to resolve this translation problem. In this chapter, I help you understand why this argument is valid without invalidating everything you've discovered about logic so far. Instead, I show you how to take all of SL and extend it to a new and more powerful formal logical language called *quantification logic* (*QL*). QL introduces you to two new symbols that enable you to express logical concepts that are way beyond the reach of SL. Get ready, because a whole new logical world awaits!

Taking a Quick Look at Quantification Logic

QL uses much of the same structure as SL. You may recall that SL has both constants and variables (see Chapter 4). A *constant* represents a statement in a natural language (such as English), whereas a *variable* represents a statement in SL. QL also has both constants and variables, but it uses them in a different way that allows you to break statements into smaller pieces for greater precision.

In some books, quantification logic is called *predicate logic* or *first-order predicate logic*.

Identifying individual constants and property constants

REMEMBER

QL has the following two types of constants (rather than just one like in SL):

- **Individual constants:** An *individual constant* represents the subject of a sentence with a lowercase letter from *a* to *u*.
- **Property constants:** A *property constant* represents the *predicate* of a sentence (the part that expresses something about the subject) with a capital letter from *A* to *Z*.

As in SL, the formal logical language QL allows you to translate statements in English into symbols. You may remember from SL that when you want to translate a statement from English into formal logic, the first step is to define some constants. For example, suppose you want to translate the following statement into QL:

David is happy.

First, define an individual constant to represent the subject of the sentence.

Let *d* = David

Next, define a property constant to represent the predicate of the sentence.

Let *Hx* = *x* is happy

To write the statement in QL, just replace the *x* with the individual constant.

Hd

WARNING

In English, the subject usually comes before the predicate. But in QL, this order is always reversed. This reversal can be confusing at first, so make sure you keep the order clear.

This example already demonstrates the added flexibility QL affords you. With SL, you'd be able to assign only one variable to the entire statement, like this:

Let *D* = David is happy. (SL TRANSLATION)

Here's another example of how to translate an English statement into QL:

> The cat was on the mat.

First, define both the individual and the property constants, like so:

> Let *c* = the cat
>
> Let *Mx* = *x* was on the mat

Now, you can translate the original statement into QL as

> *Mc*

Note that this translation works well no matter how long the subject and predicate are. For example, you can even translate the following wordy sentence:

> My obstreperous aunt from Wisconsin with three sniveling French poodles is leaving on a three-month tour of six countries in Europe and Africa.

In this case, define your constants as follows:

> Let *a* = my obstreperous aunt from Wisconsin with three sniveling French poodles
>
> Let *Lx* = *x* is leaving on a three-month tour of six countries in Europe and Africa

Then, you can easily translate the statement as

> *La*

WARNING

Use an individual constant to represent only a single thing — a constant can never be used for more than one thing.

For instance, in the previous example, the subject of the sentence is the aunt, and the poodles are just part of her description. But suppose the sentence read like this:

> My obstreperous aunt from Wisconsin *and* her three sniveling French poodles *are* leaving on a three-month tour of six countries in Europe and Africa.

In QL, you can't use one constant to represent the group of three poodles. Instead, you need to define three more individual constants — one for each of the three poodles. For example, suppose the poodles are named Fifi, Kiki, and Elvis. You may use the following definitions:

Let *f* = Fifi

Let *k* = Kiki

Let *e* = Elvis

Then, because they're all going on the tour together, you can express the statement as

$La \ \& \ Lf \ \& \ Lk \ \& \ Le$

As you can see, the and-operator from SL is also used in QL. In the next section, you see how all five SL operators come into play.

REMEMBER

Formally defining constants is an important technical point, but it's easy and you'll probably get the hang of it quickly. So, from now on, I leave out this step unless it's necessary for clarity.

Incorporating the SL operators

REMEMBER

QL contains the five operators from SL: the not-operator (~), the and-operator (&), the or-operator ($\vee$), the if-operator ($\rightarrow$), and the if-and-only-if-operator ($\leftrightarrow$). You can get a lot of mileage out of this new QL formulation for statements, especially when you add in the SL operators you already know. Suddenly, with QL, you can translate sentences into symbols at a much deeper level than you can when you're just using SL.

For example, take the statement

Nadine is an accountant, and her secretary is on vacation.

In SL, you can only crudely break the sentence into two parts.

$N \ \& \ S$ (SL TRANSLATION)

With QL, however, you can distinguish four separate pieces of the sentence and use these four pieces to represent the sentence as follows:

$An \ \& \ Vs$

Similarly, the statement

Genevieve is either a Scorpio or an Aquarius.

can be translated as

$Sg \vee Ag$

Here, the sentence is understood as "*Either* Genevieve is a Scorpio *or* Genevieve is an Aquarius."

In a similar way, the statement

If Henry is a nice guy, then I am the Queen of Sheba.

can be translated as

$Nh \rightarrow Qi$

And, finally, the statement

Five is odd if and only if it's not divisible by two.

can be translated as

$Of \leftrightarrow \sim Df$

TIP

In this sentence, the word *it* is a pronoun that stands in for *five*, so I use the individual constant *f* in both cases.

These examples demonstrate that the single-letter constants of SL are replaced by double-letter terms in QL. But never fear — you can still combine these terms according to the same rules you used in SL.

Understanding individual variables

REMEMBER

In QL, an *individual variable* represents an unspecified subject of a sentence with a lowercase letter *x*, *y*, or *z*. (In some examples, when three variables isn't enough, I also use *v* and *w*.)

In the section "Identifying individual constants and property constants" earlier in the chapter, I introduce individual variables in the formal definition of property constants. For example, in the definition

Let Hx = *x* is happy

the individual variable is *x*. And as I've already demonstrated, you can replace this variable with an individual constant to create a QL statement. For example, when you represent the sentence "David is happy" in QL as *Hd*, you replace the variable *x* with the constant *d*.

TECHNICAL STUFF

You may notice that QL has individual constants, individual variables, and property constants, but not *property variables*. Property variables come into play in *second-order predicate logic* (see Chapter 21), which is why QL is also called *first-order predicate logic*.

REMEMBER

To save space, I shorten the phrase *individual variable* to *variable* throughout Part 4 of the book.

Expressing Quantity with Two New Operators

Earlier in this chapter, I show you a few examples to demonstrate how QL represents statements that you can also express in SL. But in this section, I show you how QL goes into territory that SL *can't* cover.

The main vehicles for this new expressiveness in QL are two new symbols: $\forall$ and $\exists$.

Using the universal quantifier

REMEMBER

The symbol $\forall$ is called the *universal quantifier*. You always use this symbol in conjunction with a variable, such as x, with $\forall x$ translating into English as "For all x. . . ."

At the beginning of this chapter, you saw how troublesome it was to translate the statement "All of my children are honest" into SL in any useful way. But with the universal quantifier, you now have the power to do it.

For example, first, you need to attach the universal quantifier to a variable. You can set up the problem with empty brackets, as shown here:

$$\forall x[\ \ ]$$

After you've set this problem up, you can use the rules you already know to translate the rest of the statement. Then you place the result in the brackets, like this:

$$\forall x[Cx \rightarrow Hx]$$

You read this statement as "For all x, if x is my child, then x is honest." It may seem a bit clunky at first, but after you get used to the setup, the possibilities are literally endless.

TIP

You can also use ∀ to translate words like *every* and *each*. I get into translations in detail in Chapter 16.

Conveying existence with the existential quantifier

REMEMBER

The symbol ∃ is called the *existential quantifier*. You always use it in conjunction with a variable, such as x, with $\exists x$ translating into English as "There exists an x such that. . . ."

At the beginning of this chapter, you saw that SL is limited in its ability to express the statement "At least one of my children is a lawyer." However, the existential quantifier makes expression of this statement possible.

First, as with the universal quantifier, attach the existential quantifier to a variable.

$$\exists x[\ \ ]$$

Now, inside the brackets, translate the rest of the statement.

$$\exists x[Cx \,\&\, Lx]$$

You can read this statement as "There exists an x such that x is my child and x is a lawyer."

Similarly, you can use ∃ to translate the statement "At least one honest lawyer exists" as follows:

$$\exists x\,[Hx \,\&\, Lx]$$

This statement translates literally as "There exists an x such that x is a lawyer and x is honest."

So, here's your first argument in QL:

$$\forall x\,[Cx \rightarrow Hx],\ \exists x\,[Cx \,\&\, Lx] \therefore \exists x\ [Hx \,\&\, Lx]$$

As you can see, a lot more is going on here than in the SL version. And that's a good thing, because QL has captured in symbols exactly what's valid about this argument. But, before you look at how to prove the argument is valid, you'll need to get a little more background information.

TIP

You can also use ∃ to translate words such as *some*, *there is*, and *there are*. I discuss translations of this kind in Chapter 16.

Creating context with the domain of discourse

Because QL can handle a wider range of circumstances than SL, a new type of ambiguity arises when you're translating from English to QL. For example, look at this statement:

> Everyone is wearing a suit.

You can translate this statement into QL as follows:

$$\forall x\,[Sx]$$

This translates literally as "For all x, x is wearing a suit," which is false. But if the context you're speaking of is a meeting of four coworkers, the statement may be true.

To prevent this potential for confusion, you need to be clear about the context of a statement. For this reason, you must place variables in a context called the *domain of discourse*.

REMEMBER

The *domain of discourse* provides a context for statements that contain variables. For example, if the domain of discourse is *people*, then x can stand for me, or you, or your Aunt Henrietta, but not for Bart Simpson, your cell phone, or the planet Jupiter.

TECHNICAL STUFF

In math, the domain of discourse is often taken for granted. For example, when you write the equation $x+2=5$, you can assume that the variable x stands for a number, so the domain of discourse in this case is the set of all real numbers.

Declaring the domain of discourse

How you translate a statement from English into QL is often dependent upon the domain of discourse that you *declare* — that is, state for a given argument. To see why this dependency exists, take a look at two translations of an argument that use two different domains of discourse.

Premises:

> All of my children are honest.
>
> At least one of my children is a lawyer.

Conclusion:

> At least one honest lawyer exists.

For the first translation, here's a declaration I use:

Domain: Unrestricted

REMEMBER

An unrestricted domain means that x can be literally anything: a flea, a unicorn, the Declaration of Independence, the Andromeda Galaxy, the word *butter*, or your first kiss. (You thought I was kidding when I said *anything*, didn't you?) This wide range isn't a problem, though, because the statements themselves put restrictions on what x can be.

With this domain of discourse, the argument translates into QL as

$$\forall x\,[Cx \rightarrow Hx],\ \exists x\,[Cx \,\&\, Lx] : \exists x\,[Hx \,\&\, Lx]$$

Choosing the domain of discourse wisely

When writing an argument, you can use the domain of discourse to your advantage by cleverly choosing a domain that makes the translation from English into QL easier and more concise.

For this second translation of the same argument, I can start out by declaring a different domain.

Domain: My children

Now, all the variables are under a significant restriction, which means I can translate the statement "All of my children are honest" as

$$\forall x\,[Hx]$$

instead of the earlier statement I used:

$$\forall x\,[Cx \rightarrow Hx]$$

You read $\forall x\,[Hx]$ as "For all *x*, *x* is honest" or simply "All *x* is honest." Note that the statement itself doesn't mention the fact that x is one of my children (Cx from the previous statement) because the domain of discourse handles this restriction.

Similarly, using the same domain of discourse, you can translate the statement "At least one of my children is a lawyer" in the following way:

$$\exists x\,[Lx]$$

You read this statement as "There exists an x such that x is a lawyer." Again, the statement itself doesn't mention that x is one of my children because this restriction is covered by the domain of discourse.

Finally, you can translate "At least one honest lawyer exists" as

$$\forall x\,[Hx \,\&\, Lx]$$

This is the same translation as when the domain is unrestricted because the statement contains no mention of my children. Thus, within the domain *My children*, the same argument translates as

$$\forall x\,[Hx],\ \exists x\,[Lx] \therefore \exists x\,[Hx \,\&\, Lx]$$

As you can see, this argument is a little shorter and simpler than the argument using an unrestricted domain.

WARNING

Be sure to use just *one* domain of discourse for a single argument or for any statements you're analyzing as a group. Switching between domains is a lot like trying to compare apples and oranges, and the result is usually a mixed-up argument that doesn't give you proper results.

REMEMBER

The domain of discourse is important from a technical standpoint, but most of the time you can use an unrestricted domain. So, from now on — unless I state otherwise — assume that the domain is unrestricted.

Picking Out Statements and Statement Forms

In Chapter 5, I describe the difference in SL between a statement — for example, $\sim(P \to Q)$ — and a statement form — such as $\sim(x \to y)$. Simply put, a *statement* has constants but no variables, and a *statement form* has variables but no constants. (I talk more about this difference in the section "Knowing the difference between statements and statement forms" later in this chapter.)

In QL, however, a single expression can have both constants and variables. In this section, I explain how you use constants and variables, and show you what constitutes a statement in QL.

Determining the scope of a quantifier

In Chapter 5, I discuss how parentheses limit the scope of an SL operator. Consider, for example, this SL statement:

$$P \,\&\, (Q \vee R)$$

The and-operator in this statement is *outside* the parentheses, so its scope is the whole statement, which means this operator affects every variable in the statement. On the other hand, the or-operator is *inside* the parentheses, so its scope is limited to what's inside the parentheses. That is, this operator affects only the constants Q and R, but not P.

REMEMBER

In QL, the scope of the quantifiers $\forall$ and $\exists$ is limited in a similar way, but with brackets instead of parentheses. For example, check out this statement:

$$\forall x\,[Cx \rightarrow Nx] \,\&\, Rb$$

The scope of the quantifier $\forall$ is limited to what's inside the brackets: $Cx \rightarrow Nx$. In other words, this quantifier doesn't affect the and-operator or the substatement Rb.

Discovering bound variables and free variables

You thought you knew enough about variables already, didn't you? Well, get ready, because here's another exciting lesson: When you're sitting at your desk pondering an expression, remember that every variable in that expression is considered either *bound* or *free.* Here's what I mean:

- A *bound variable* is a variable that's being quantified.
- A *free variable* is a variable that isn't being quantified.

Easy enough, right? To completely understand the difference between bound variables and free variables, take a look at the following expression:

$$\exists x\,[Dx \rightarrow My] \,\&\, Lx$$

Can you guess which of the variables in this expression are bound and which are free?

REMEMBER

Here are the important points to know about free and bound variables:

- **When a variable is *outside* all brackets, it's always free.** This distinction happens because the scope of a quantifier is always limited to what's *inside* a set of brackets. So, the x in Lx is a free variable.
- **When a variable is inside a set of brackets, it isn't necessarily bound.** In the example, the variable y is also free even though it's inside the brackets. This is because these brackets define the scope of $\exists x$, which is operating on x, not y.

  You can probably guess by now that the variable x in Dx is a bound variable. You can be sure of this because it appears inside the brackets and because these brackets define the scope of $\exists x$, which does quantify x.

TIP

The difference between bound variables and free variables becomes important in Chapter 17 when you dive into QL proofs. For now, make sure you understand that a bound variable is being quantified and a free variable isn't being quantified.

Knowing the difference between statements and statement forms

When you understand the information about bound variables and free variables from the previous section, you can easily tell the difference between statements and statement forms.

REMEMBER

When you're evaluating any logical expression, you can be sure that each one falls into one of these two categories:

- **Statements:** A *statement* contains no free variables.
- **Statement forms:** A *statement form* contains at least one free variable.

Distinguishing a statement from a statement form is that easy. For example, consider the following expression:

$$\forall x\,[Cx \rightarrow Hx]\,\&\,Ok$$

This expression is a statement because it has no free variables. Note that both instances of the variable x appear inside the brackets within the scope of the quantifier $\forall x$. Additionally, k appears outside the brackets, but it's a constant, not a variable.

On the other hand, consider this expression:

$$\forall x\,[Cx \rightarrow Hx]\,\&\,Ox$$

This expression has one free variable (the x in Ox), which falls outside the brackets and is therefore outside the scope of the quantifier $\forall x$. So, the expression is a statement form rather than a statement.

Formal definitions in QL use statement forms that you can then turn into statements. For example, look at this definition:

> Let Ix = x is Italian

The expression Ix is a statement form because the variable x is free. But you turn this statement form into a statement when you replace the variable with an individual constant. For example, to translate the statement

> Anna is Italian.

you can write

> Ia

Similarly, binding the variable in a statement form turns the statement form into a statement. For example, to translate the statement

> Somebody is Italian.

you can write

> $\exists x\,[Ix]$

The difference between statements and statement forms becomes important in Chapter 17 when you begin writing QL proofs. For now, just make sure you understand that a statement has no free variables.

IN THIS CHAPTER

- Recognizing the four basic forms of categorical statements
- Using the universal or existential quantifier to translate the basic forms
- Translating English statements that begin with words other than *all*, *some*, *not all*, or *no*

Chapter 16
Embracing QL Translations

In Chapter 15, I introduce you to quantification logic (QL), paying special attention to the two quantifiers $\forall$ and $\exists$ (the universal and existential quantifiers, respectively). I also show you how to translate a few simple statements into QL.

In this chapter, I show you how to translate the four basic forms of *categorical statements* from English into QL. These tend to be statements that begin with the words *all*, *some*, *not all*, and *no*. (Take a quick look at Chapter 2 for more information about categorical statements.)

This chapter also covers how to translate each of these four forms using either the $\forall$ or the $\exists$ quantifier. Finally, I show you how to recognize categorical statements that don't begin with any of the four keywords.

Translating the Four Basic Forms of Categorical Statements

In this section, I explain how to translate the words *all, some*, *not all,* and *no.* You'll use these words a lot while figuring out how to translate categorical statements from English into QL. (If you need to, take a peek at Chapter 2 to make sure you understand the basic forms of categorical statements.) Then I show you how to translate statements beginning with *all* and *some*. After you understand how to translate those statements, the statements beginning with *not all* and *no* are pretty easy.

When you know how to translate statements beginning with the words *all*, *some*, *not all*, and *no*, much of the hard work of translating English into QL is done. In general, the easiest way to approach these types of statements is to set up a quantifier and brackets, as shown in Table 16-1.

TABLE 16-1 Translations of the Four Basic Forms of Categorical Statements

English Word	QL Quantifier
All	$\forall$ [ ]
Some	$\exists$ [ ]
Not all	$\sim\forall x$ [ ]
No	$\sim\exists x$ [ ]

Working with "all" and "some"

Translating statements from English into QL isn't difficult, but you have to be aware of a couple sticking points regarding restricted and unrestricted *domains of discourse* (the context for statements that contain variables). (For a refresher on how to use the domain of discourse, flip to Chapter 15.)

In this section, I give you an overview of how to translate statements beginning with the words *all* and *some* into QL.

Using a restricted domain

Two statements in English may be identical except for the words *all* or *some*. For this reason, you may expect their QL translations to be identical except for the choice of quantifier (∀ or ∃). When the domain is restricted, this is often exactly how things work out.

For example, take the following two statements:

> All horses are brown.
>
> Some horses are brown.

Suppose you use a restricted domain of discourse.

> Domain: Horses

If you use a restricted domain, you need to define only one property constant to translate the two statements. For example:

> Let Bx = x is brown

Now you can translate both statements as follows:

> $\forall x\,[Bx]$
>
> $\exists x\,[Bx]$

As you can see, the only difference between the two QL statements is that you replaced ∀ in the first statement with ∃ in the second. Just as you'd expect, nothing tricky here.

For the rest of this chapter, I focus on translations in an unrestricted domain.

Using an unrestricted domain

Restricting the domain isn't always possible, so this time, suppose your domain of discourse is unrestricted.

> Domain: Unrestricted

Now, to translate the English statements from the previous section, you need another property constant. So, here are the two new constants you need, plus the one from the previous section:

Let Bx = x is brown

Let Hx = x is a horse

Let Fx = x can fly

Now you can translate the statement "All horses are brown" as

$$\forall x\,[Hx \rightarrow Bx]$$

This statement translates literally as "For all x, if x is a horse, then x is brown."

On the other hand, you can translate the statement "Some horses are brown" as

$$\exists x\,[Hx \,\&\, Bx]$$

This statement translates literally as "There exists an x such that x is a horse and x is brown."

Notice that the difference between the two statements is more than just the quantifiers $\forall$ and $\exists$. In the first statement, you use an if-operator, whereas in the second, you use an and-operator.

Take a look at what happens, though, if you mix up these operators. Suppose you tried to translate the first statement as

$$\forall x\,[Hx \,\&\, Bx] \qquad \text{WRONG!}$$

This translation is wrong because what it actually says is "For all x, x is both a horse and brown." More informally, it says, "Everything is a brown horse." And because the domain is unrestricted, everything really does mean *everything*, which obviously isn't what you mean.

As another example, suppose you tried to translate the statement

Some horses can fly.

as

$$\exists x\,[Hx \rightarrow Fx] \qquad \text{WRONG!}$$

Again, this translation is wrong because it says, "There exists an *x* such that if *x* is a horse, then *x* can fly." The problem here is a bit more subtle.

Suppose *x* is something other than a horse — for example, a shoe. Then, the first part of the statement inside the brackets is false, which makes everything inside the brackets true. So, in this case, the statement seems to be telling you that the existence of a shoe means that some horses can fly. Again, something has been lost in translation.

REMEMBER

When translating from English into QL, keep the following general rules in mind:

- When using $\forall$, use an if-statement inside the brackets.
- When using $\exists$, use an and-statement inside the brackets.

Moving on to "not all" and "no"

After you know how to translate the two positive forms of categorical statements, translating the two negative forms — *not all* and *no* — is easy. As I discuss in Chapter 2, a *not all* statement is just the contradictory form of an *all* statement. Similarly, a *no* statement is the contradictory form of a *some* statement.

For example, you already know how to translate the statement

All horses are brown.

as

$$\forall x[Hx \rightarrow Bx]$$

Thus, its contradictory form is

Not all horses are brown.

This statement translates easily as follows:

$$\sim \forall x[Hx \rightarrow Bx]$$

Furthermore, because *all* and *not all* statements are contradictory, you know that exactly one of these statements is true and the other is false.

You also know how to translate the statement

Some horses can fly.

as

$\exists x\,[Hx \,\&\, Fx]$

This statement's contradictory form is

No horses can fly.

As you may have guessed, this statement translates as

$\sim\exists x\,[Hx \,\&\, Fx]$

Again, because *some* and *no* statements are contradictory, exactly one is true and the other is false.

Discovering Alternative Translations of the Basic Forms

Because QL is very versatile, it offers you more than one way to translate each form of the categorical statements. In this section, I show you how to translate each of the four basic forms of categorical statements using both the $\forall$ and $\exists$ quantifiers.

REMEMBER

Knowing both versions of each translation will help you understand the hidden connection between the universal and existential quantifiers ($\forall$ and $\exists$).

For the examples in this section, I use the following definitions:

Let Dx = x is a dog

Let Px = x is playful

Table 16-2 organizes this section's information about translating the four basic forms of categorical statements from English into QL. For each form, I list two English statements and two QL translations. The first statement gives you the simplest way to say it in English and its most direct translation into QL. The second gives you an alternative phrasing of the same idea with a translation of this phrasing.

TABLE 16-2 Alternative Translations of the Four Basic Forms of Categorical Statements

English Translations	QL Translations
All dogs are playful.	$\forall x\,[Dx \rightarrow Px]$
(No dogs are not playful.)	$(\sim\exists x\,[Dx\ \&\ \sim Px])$
Some dogs are playful.	$\exists x[Dx\ \&\ Px]$
(Not all dogs are not playful.)	$(\sim\forall x\,[Dx \rightarrow \sim Px])$
Not all dogs are playful.	$\sim\forall x\,[Dx \rightarrow Px]$
(Some dogs are not playful.)	$(\exists x\,[Dx\ \&\ \sim Px])$
No dogs are playful.	$\sim\exists x\,[Dx\ \&\ Px]$
(All dogs are not playful.)	$(\forall x\,[Dx \rightarrow \sim Px])$

Translating "all" with ∃

It makes perfect sense to translate the statement "All dogs are playful" as

$$\forall x\,[Dx \rightarrow Px]$$

TIP

Think about it. Even if it's upside down, the ∀ stands for *all*. That's why I discuss this translation first.

But, when you think about it some more, you may realize that when you say all dogs are playful, you're making this equivalent statement:

> No dogs are *not* playful.

In other words, this statement is just a *no* statement with the word *not* inserted. So, a perfectly good translation is

$$\sim\exists\,[Dx\ \&\ \sim Px]$$

The literal translation is this: "It isn't true that there exists an *x* such that *x* is a dog and *x* is not playful." Both English statements mean the same thing, so both QL statements are *semantically equivalent*. (For more information about semantic equivalence, see Chapter 6.)

Interpreting "some" with ∀

In the same way that you can translate *all* with ∃, you can use the ∀ quantifier to translate the word *some*. For example, consider the statement

> Some dogs are playful.

The direct translation of this statement into QL is

$$\exists x\,[Dx \,\&\, Px]$$

Note that this statement means the same thing as

> Not all dogs are *not* playful.

So, you can treat this statement as a *not all* statement with the word *not* inserted, which translates as

$$\sim \forall x\,[Dx \rightarrow \sim Px]$$

In this case, the literal translation is "It isn't true that for all *x*, if *x* is a dog, then *x* is not playful." Because the two statements in English mean the same thing, the two QL statements are semantically equivalent.

Converting "not all" with ∃

Suppose you want to translate the following statement:

> Not all dogs are playful.

You already know how to translate this statement as

$$\sim \forall x\,[Dx \rightarrow Px]$$

But here's another way to think about it: Because not all dogs are playful, *there's at least one* dog somewhere that is *not* playful. So, you can rephrase the original statement as

> Some dogs are not playful.

To express this idea, you can use the following translation:

$$\exists x\,[Dx \,\&\, \sim Px]$$

The more exact translation for this statement is "There exists an *x* such that *x* is a dog and *x* is not playful." Again, these two English statements mean the same thing, and so do their counterparts in QL.

Transforming "no" with ∀

Suppose you want to translate this statement:

> No dogs are playful.

The simplest translation is

$$\sim\exists x\,[Dx \,\&\, Px]$$

The more exact translation is "It is not true that there exists an *x* such that *x* is a dog and *x* is playful."

As you may have guessed, you can think of this translation another way. For instance, the fact that no dogs are playful means the following:

> All dogs are not playful.

You can express this idea in QL as

$$\forall x\,[Dx \rightarrow \sim Px]$$

The more exact translation in this case is: "For all *x*, if *x* is a dog, then *x* is not playful." Once again, because the two English statements mean the same thing, their translations into QL are semantically equivalent.

Identifying Statements in Disguise

In the previous sections of this chapter, you see how to translate the four basic forms of categorical statements from English into QL. Well, you're in for a treat, because in this section, I help you broaden your understanding to go beyond the basics.

Here, I show you how to recognize statements that don't start with the exact words you're used to translating. Then you can take on a much wider variety of statements in English and turn them into QL.

Distinguishing "all" statements

REMEMBER

Even if a statement doesn't begin with the word *all*, it may mean something close enough to use the ∀ quantifier. Here are some examples of this type of English statement, followed by translations into QL:

Any father would risk his life to save his child.

$\forall x\,[Fx \rightarrow Rx]$

Every man in this room is single and eligible.

$\forall x\,[Mx \rightarrow (Sx \,\&\, Ex)]$

The family that prays together stays together, while every family that takes vacations together always laughs a lot.

$\forall x\,[Px \rightarrow Sx] \,\&\, \forall x\,[Vx \rightarrow Lx]$

REMEMBER

In the last example, you can quantify the variable *x* twice because you're making a statement about *all* families.

Spotting "some" statements

You may run across certain statements that are close enough to *some* statements that you'll want to use the ∃ quantifier to translate them. For example, consider these statements with their translations:

At least one of the guests at this party committed the crime.

$\exists x\,[Gx \,\&\, Cx]$

Many teenagers are rebellious and headstrong.

$\exists x\,[Tx \,\&\, (Rx \,\&\, Hx)]$

Both corrupt plumbers and honest judges exist.

$\exists x\,[Px \,\&\, Cx] \,\&\, \exists y\,[Jy \,\&\, Hy]$

REMEMBER

Notice in the last example that you need two variables, x and y. These variables make the relationship clear — that is, while both groups (corrupt plumbers and honest judges) exist, they don't necessarily overlap.

Noticing "not all" statements

You can easily translate a few statements that don't include the words *not all* by using $\sim \forall$. For example, consider these statements that are considered *not all* statements:

Not every musician is flaky.

$\sim \forall x\,[Mx \rightarrow Fx]$

Fewer than 100 percent of accountants are reserved and thorough.

$\sim \forall x\,[Ax \rightarrow (Rx \,\&\, Tx)]$

All that glitters is not gold.

$\sim \forall x\,[Ix \rightarrow Ox]$ (Because the words *glitters* and *gold* both begin with the letter *g*, I use the first vowel in each word.)

WARNING

Even though this last example begins with the word *all*, it's a *not all* statement that means "Not all things that glitter are gold." So, be sure to think through what a statement is really saying before translating it!

Recognizing "no" statements

You've probably guessed by now that some statements that don't start with the word *no* are easy to translate using $\sim \exists$. You're right on the money. Consider these statements and their translations:

Not a single person on the jury voted to acquit the defendant.

$\sim \exists x\,[Jx \,\&\, Vx]$

There aren't any doctors or even college graduates in our family.

$\sim \exists x\,[Fx \,\&\, (Dx \vee Cx)]$

Nobody who buys a restaurant makes money without putting in long hours.

$\sim \exists x\,[Bx \,\&\, (Mx \,\&\, \sim Px)]$

You can think of the last statement as "*No* person exists who buys a restaurant *and* makes money *and* doesn't put in long hours."

IN THIS CHAPTER

- Comparing proofs in SL and QL
- Applying quantifier negation in QL
- Writing QL proofs using the four quantifier rules: UI, EI, UG, and EG

Chapter 17
Proving Arguments with QL

Here's the good news: If you've got the hang of sentential logic (SL) proofs, you already know 80 percent of what you need to write proofs in quantification logic (QL). So, first you should figure out how much you already grasp about proof writing. If you want a refresher, take a look at the chapters in Part 3, which tell you everything you need to know.

In this chapter, I illustrate how proofs in QL are similar to those in SL. One way they're similar is that they both use the eight implication rules (see Chapter 9) and the ten equivalence rules (refer to Chapter 10). I explain exactly how and when to use these rules.

After you get comfortable with a few simple proofs in QL, I introduce an additional rule, *quantifier negation* (**QN**), which allows you to make changes that affect the universal or existential quantifiers ($\forall$ or $\exists$). Luckily, this rule, which is the first rule that's unique to QL, is an easy one to master.

The rest of the chapter focuses on the four quantifier rules: *universal instantiation* (**UI**), *existential instantiation* (**EI**), *universal generalization* (**UG**), and *existential generalization* (**EG**). The first two rules allow you to remove each type of quantifier from a QL statement. The other two rules permit you to add either type of quantifier to a QL statement. As a group, the four quantifier rules allow you to use the eight implication rules from SL to their maximum benefit.

Applying SL Rules in QL

The 18 rules of inference from SL are a huge slice of what you need to know about writing proofs in QL. These rules include the eight implication rules from Chapter 9 and the ten equivalence rules from Chapter 10. In many cases, you can transfer these rules over to QL proofs with only minor adjustments.

This section highlights the many similarities between proofs in QL and SL.

Comparing similar SL and QL statements

One key difference between SL and QL is the way the two languages handle simple statements. For example, to translate the statement "Howard is sleeping" into SL, you can write

H	(SL TRANSLATION)

To translate the same statement into QL, you can write

Sh	(QL TRANSLATION)

Similarly, to translate the more complex statement "Howard is sleeping and Emma is awake" into SL, you can write

H & *E*	(SL TRANSLATION)

To translate the same statement into QL, you can write

Sh & *Ae*	(QL TRANSLATION)

REMEMBER

Neither of these English statements contains any words that require you to use a universal or existential quantifier (∀ or ∃). That's why you can translate them both into either SL or QL.

In fact, in such cases only one difference exists between SL and QL translations: the constants. For example, in SL, you translate a simple English statement into a single-letter constant (for example, *H* or *E*), which can then stand on its own as an SL statement.

In QL, however, you translate the same English statement into a two-letter combination of a property constant and an individual constant (for example, *Sh* or *Ae*), which can then stand on its own as a QL statement. (Flip to Chapter 4 for more on SL constants and Chapter 15 for QL constants.)

TIP

For QL statements, the general rule is just to treat the two-letter combinations as *indivisible* building blocks of larger statements. That is, you're usually not going to break apart the property and individual constants, so you can just treat them as units, as you would treat single-letter constants in SL. (See Chapter 18 to find out about identities, the one exception to this rule.)

Transferring the eight implication rules from SL into QL

REMEMBER

Because SL and QL are so similar, you can easily apply the eight implication rules to any QL statement or statement form, just as you would in SL.

WARNING

As in SL, when you use the eight implication rules in QL, you can apply them only to whole statements or whole statement forms — never to partial statements or partial statement forms. (Check out Chapter 9 for a refresher on how to use the implication rules.)

Working with QL statements without a quantifier

The first thing to know is how the eight implication rules apply to QL statements without a quantifier. For example, suppose you want to prove this argument:

$$Jn \rightarrow Bn,\ {\sim}Bn \vee Ep,\ Jn\ \therefore\ Ep$$

Begin, as always, by listing the premises.

1. $Jn \rightarrow Bn$	**P**
2. ${\sim}Bn \vee Ep$	**P**
3. Jn	**P**

Now, continue by using *modus ponens* (**MP**) and *disjunctive syllogism* (**DS**) as you would with an SL proof, but treat the two-letter constants as indivisible chunks, just like the single-letter constants and variables in QL.

4. Bn	1,3 **MP**
5. Ep	2,4 **DS**

Proving QL statements that include a quantifier

You can also use the eight implication rules from SL to prove QL statements with one or more quantifiers. As I note earlier in the chapter, just make sure you're applying the rules to whole statements and statement forms.

For example, suppose you want to prove the following argument:

$$\forall x\,[Nx] \,\&\, \exists y\,[Py] : (\forall x\,[Nx] \vee \forall y\,[Py]) \,\&\, (\exists x\,[Nx] \vee \exists y\,[Py])$$

Again, start with the premise.

1. $\forall x\,[Nx] \,\&\, \exists y\,[Py]$	**P**

Now you can work with this entire statement using the implication rules as you would in SL proofs. Begin by using *simplification* (**Simp**) to unpack the two sub-statements within the and-statement.

2. $\forall x\,[Nx]$	1 **Simp**
3. $\exists y\,[Py]$	1 **Simp**

Next, use *addition* (**Add**) to build two new or-statements.

4. $\forall x\,[Nx] \vee \forall y\,[Py]$	2 **Add**
5. $\exists x\,[Nx] \vee \exists y\,[Py]$	3 **Add**

Finally, use *conjunction* (**Conj**) to build a new and-statement.

6. $(\forall x\,[Nx] \vee \forall y\,[Py]) \,\&\, (\exists x\,[Nx] \vee \exists y\,[Py])$	4,5 **Conj**

WARNING

As in SL, you can't apply the implication rules to part of a statement. In QL, this holds true even when that part is the only thing in the brackets of a statement with a quantifier.

For example, here's an *invalid* argument:

Premises:

Some members of my family are doctors.

Some members of my family never finished high school.

Conclusion:

Some doctors never finished high school.

And here's the "proof" of this argument:

$$\exists x\,[Fx \,\&\, Dx], \exists x\,[Fx \,\&\, Nx] : \exists x\,[Dx \,\&\, Nx]$$

1. $\exists x\,[Fx \,\&\, Dx]$	**P**
2. $\exists x\,[Fx \,\&\, Nx]$	**P**
3. $\exists x\,[Dx]$	1 **Simp** WRONG!
4. $\exists x\,[Nx]$	2 **Simp** WRONG!
5. $\exists x\,[Dx \,\&\, Nx]$	3,4 **Conj** WRONG!

Obviously, something is very wrong here. The moral of the story is that you aren't allowed to apply **Simp**, **Conj**, or any other implication rule to the stuff inside the brackets and just ignore the rest of the statement. (Later in this chapter, I show you how to prove arguments without breaking this rule.)

Working with QL statement forms

In Chapter 15, I show you that in QL, a property constant combined with an individual variable (for example, Px) isn't really a statement unless it's inside a pair of brackets and modified by a quantifier (for example, $\forall x[Px]$). When the combined expression appears free, it isn't a statement at all, but rather a *statement form.* (To recall the distinction between free and bound variables, and how they relate to statements and statement forms, see Chapter 15.)

The good news is that when writing proofs, you can treat statement forms the same as statements. Later in the chapter, I give specific examples of how statement forms can arise in the course of a QL proof. For now, just know that the eight implication rules also apply to statement forms.

Employing the ten SL equivalence rules in QL

REMEMBER

As in SL, you can apply the ten equivalence rules to any *whole* QL statement or statement form or to any *part* of a QL statement or statement form.

In Chapter 10, I explain that the equivalence rules give you greater flexibility than the implication rules. One reason for this flexibility is that you can apply the ten equivalence rules not only to whole SL statements but also to parts of statements. This same application works in QL. For example, suppose you want to prove the following argument:

$$\exists x\,[Cx \rightarrow {\sim}Sx] \therefore \exists x\,{\sim}[Cx \,\&\, {\sim}Sx]$$

1. $\exists x\,[Cx \rightarrow {\sim}Sx]$	**P**

Because the $\exists x$ remains unchanged from the premise to the conclusion, your main challenge here is to change the rest of the statement. And you can do this by using two of your favorite equivalence rules, *implication* (**Impl**) and *De Morgan's theorem* (**DeM**). (Note that as you apply **DeM**, the brackets in a QL statement function just as parentheses do in SL.)

2. $\exists x\,[{\sim}Cx \vee {\sim}Sx]$	1 **Impl**
3. $\exists x\,{\sim}[Cx \,\&\, Sx]$	2 **DeM**

This flexibility with regard to equivalence rules applies equally well to statement forms.

For example, the expression $Cx \rightarrow {\sim}Sx$ by itself is a statement form because the variable x is unquantified (see Chapter 15). You can use the equivalence rule **Impl** (see Chapter 10) to rewrite this statement form as ${\sim}Cx \vee {\sim}Sx$.

Later in the chapter, I discuss how statement forms find their way into QL proofs.

Transforming Statements with Quantifier Negation

In transferring the rules for SL proofs over to QL, you've had to work around the quantifiers $\forall$ and $\exists$. For some proofs, working around the quantifiers in this way won't hold you back. But suppose you want to prove the following argument:

$$\forall x\,[Hx \rightarrow Bx] \therefore {\sim}\exists x\,[Hx \,\&\, {\sim}Bx]$$

This argument may look familiar if you've read Chapter 16. There, I use the premise statement to stand for "All horses are brown" and the conclusion statement to

stand for "No horses are not brown." Because these two statements are equivalent, you can start with the first and prove the second.

The problem here is that the quantifier changes from $\forall$ to $\sim\exists$, and nothing from SL is going to help with that. Luckily, **QN** comes to the rescue. Read on to see how.

Introducing QN

REMEMBER

Quantifier negation (**QN**) allows you to change a QL statement to an equivalent statement by taking the following three steps:

1. **Place a not-operator immediately before the quantifier.**
2. **Change the quantifier (from $\forall$ to $\exists$, or from $\exists$ to $\forall$).**
3. **Place a not-operator immediately after the quantifier.**

As in SL, if the resulting statement has any instances of two adjacent not-operators, you can remove both of them. This removal uses the *double negation* (**DN**) rule without referencing it, as I discuss in Chapter 10.

Table 17-1 lists the four distinct cases in which you can use **QN**.

TABLE 17-1 The Four QN Rules

	Direct Statement	Equivalent Statement
All	$\forall x\,[Px]$	$\sim\exists x\sim[Px]$
Not all	$\sim\forall x\,[Px]$	$\exists x\sim[Px]$
Some	$\exists x[Px]$	$\sim\forall x\sim[Px]$
No	$\sim\exists x\,[Px]$	$\forall x\sim[Px]$

Note that in each case, the not-operator that follows the quantifier doesn't change what's inside the brackets but simply negates the entire contents.

REMEMBER

Like the ten equivalence rules, **QN** works in both directions.

For example, you can start with this statement:

$$\sim\exists x\sim[Gx\vee Hx]$$

and end up with this one:

$$\forall x\,[Gx \vee Hx]$$

REMEMBER

And just like with the ten equivalence rules, you can apply **QN** to part of a statement.

For example, you can use **QN** to change just part of the following statement:

$$\forall x\,[Mx \vee Lx] \,\&\, \exists x\,[Cx \vee Fx]$$

to get this statement with only the first part (before the and-operator) changed:

$$\sim\exists x\,\sim[Mx \vee Lx] \,\&\, \exists x\,[Cx \vee Fx]$$

Using QN in proofs

With **QN** and the rules from SL, you can now prove the following argument:

$$\forall x\,[Bx \rightarrow Cx] \therefore \sim\exists x\,[Bx \,\&\, \sim Cx]$$

1. $\forall x\,[Bx \rightarrow Cx]$	**P**

QN handles the big change involving the quantifier.

2. $\sim\exists x\,\sim[Bx \rightarrow Cx]$	1 **QN**

The rest of the proof is just a couple tweaks using the equivalence rules.

3. $\sim\exists x\,\sim[\sim Bx \vee Cx]$	2 **Impl**
4. $\sim\exists x\,[Bx \,\&\, \sim Cx]$	3 **DeM**

TIP

Because **QN** and the equivalence rules are all true in both directions, you can just as easily prove $\sim\exists x[Bx \,\&\, \sim Cx] \therefore \forall x[Bx \rightarrow Cx]$ by reversing the steps. This reversal provides rigorous proof that these two statements mean the same thing — that is, they're *semantically equivalent* (for more information on semantic equivalence, see Chapter 6).

Table 17-2 encapsulates the information about equivalent ways to write QL expressions for *all* and *not all* statements.

TABLE 17-2 Four Equivalent Ways to Write *All* and *Not All* Statements

	All	Not All
Direct statement	$\forall x\,[Bx \rightarrow Cx]$	$\sim\forall x\,[Bx \rightarrow Cx]$
Apply **QN**	$\sim\exists x\sim[Bx \rightarrow Cx]$	$\exists x\sim[Bx \rightarrow Cx]$
Apply **Impl**	$\sim\exists x\sim[\sim Bx \vee Cx]$	$\exists x\sim[\sim Bx \vee Cx]$
Apply **DeM**	$\sim\exists x\,[Bx\,\&\sim Cx]$	$\exists x\,[Bx\,\&\sim Cx]$

Table 17-3 lists the equivalent ways to write *some* and *no* statements.

TABLE 17-3 Four Equivalent Ways to Write *Some* and *No* Statements

	Some	No
Direct statement	$\exists x\,[Bx\,\&\,Cx]$	$\sim\exists x\,[Bx\,\&\,Cx]$
Apply **QN**	$\sim\forall x\sim[Bx\,\&\,Cx]$	$\forall x\sim[Bx\,\&\,Cx]$
Apply **DeM**	$\sim\forall x\,[\sim Bx \vee \sim Cx]$	$\forall x\,[\sim Bx \vee \sim Cx]$
Apply **Impl**	$\sim\forall x[Bx \rightarrow \sim Cx]$	$\forall x[Bx \rightarrow \sim Cx]$

Exploring the Four Quantifier Rules

When they start using the rules for proofs in SL (see Chapter 9), most students find that different rules are helpful for different things. For example, **Simp** and **DS** are good for breaking statements down, while **Conj** and **Add** are good for building them up.

REMEMBER

A similar idea applies in QL. Two of the four quantifier rules are *instantiation rules*, as Table 17-4 shows. These two rules allow you to remove the quantifier and brackets from a QL statement, breaking the statement down so the SL rules can do their thing. The other two quantifier rules are *generalization rules* (also in Table 17-4). These rules allow you to add quantifiers and brackets to build up the statements you need to complete the proof.

Two of these rules — **UI** and **EG** — are relatively easy, so I go over them first. **UI** allows you to break down statements quantified with $\forall$, whereas **EG** allows you to build up statements using the $\exists$ quantifier.

TABLE 17-4 The Four Quantifier Rules in QL and Their Limitations

Quantifier	Breaking Down	Building Up
∀	***Universal Instantiation*** **(UI)** * Changes a bound variable to either a free variable or a constant.	***Universal Generalization*** **(UG)** * Changes a free variable (not a constant) to a bound variable. * This bound variable must not already appear in an earlier line in the proof justified by **EI** or an undischarged **AP**.
∃	***Existential Instantiation*** **(EI)** * Changes a bound variable to a free variable (not a constant). *This variable must not be free in an earlier line in the proof.	***Existential Generalization*** **(EG)** * Changes either a free variable or a constant to a bound variable.

After you get the hang of **UI** and **EG**, you can move on to **EI** and **UG**. These two rules are a little trickier because they include some limitations that **UI** and **EG** don't have. But, in principle, **EI** is just another breaking-down rule, and **UG** is just another building-up rule. The following sections tell you everything you need to know to figure out how to use these four rules.

Easy rule #1: Universal instantiation

REMEMBER

Universal instantiation (**UI**) allows you to do the following:

- Free the bound variable in a universal-statement by removing the quantifier and the brackets.
- Having freed this variable, uniformly change it wherever it appears to any individual constant or variable if you choose to do so.

For example, suppose you know that this statement is true:

All snakes are reptiles.

In QL, you can express this statement as

$$\forall x\,[Sx \rightarrow Rx]$$

Because you know a fact about *all* snakes, it's safe to say that a similar statement about a *specific* snake is also true.

If Binky is a snake, then Binky is a reptile.

UI allows you to make this leap formally, like this:

$Sb \rightarrow Rb$

The result is a statement that looks a lot like an SL statement, which means you can use the 18 SL rules.

Warming up with a proof

This section gives you a warm-up proof to practice with. Suppose you want to prove the validity of this argument:

Premises:

All elephants are gray.

Tiny is an elephant.

Conclusion:

Tiny is gray.

Translate the argument into QL as follows:

$$\forall x\,[Ex \rightarrow Gx], Et :\ Gt$$

As with an SL proof, begin by listing the premises.

1. $\forall x\,[Ex \rightarrow Gx]$	**P**
2. Et	**P**

Now, "unpack" the first premise using **UI**:

3. $Et \rightarrow Gt$	1 **UI**

With **UI**, I can remove the $\forall x$ and the brackets. I can then change the variable x uniformly (that is, everywhere it appears in the statement) to the individual constant t. I chose t in this case because this is the constant that appears in line 2, which becomes useful in the next step.

Now you can complete the proof using an old familiar rule from SL.

4. Gt	2,3 **MP**

Valid and invalid uses of UI

UI gives you a lot of options when you're scrounging for the next steps in a proof.

For example, suppose you're given the following premise:

$$\forall x\,[(Pa \,\&\, Qx) \rightarrow (Rx \,\&\, Sb)]$$

The simplest thing that **UI** allows you to do is simply remove the quantifier and brackets and keep the same variable.

$$(Pa \,\&\, Qx) \rightarrow (Rx \,\&\, Sb)$$

UI also allows you to change the variable — in this case, I change x to y — as long as you make this change uniformly throughout the entire expression.

$$(Pa \,\&\, Qy) \rightarrow (Ry \,\&\, Sb)$$

Furthermore, with **UI** you can change the variable to any constant you choose, even constants that already appear in the statement, as long as you make the change uniformly throughout the statement. For example:

$$(Pa \,\&\, Qa) \rightarrow (Ra \,\&\, Sb)$$
$$(Pa \,\&\, Qb) \rightarrow (Rb \,\&\, Sb)$$
$$(Pa \,\&\, Qc) \rightarrow (Rc \,\&\, Sb)$$

In each case, I've replaced the starting variable x uniformly with a constant — first a, then b, and finally c.

REMEMBER

When using **UI**, you must replace the variable uniformly with a single choice of a constant or variable. And you must preserve constants from the original statement.

For example, here's an *invalid* use of **UI** in the statement $(Pa \,\&\, Qx) \rightarrow (Rx \,\&\, Sb)$:

$(Pa \,\&\, Qa) \rightarrow (Rb \,\&\, Sb)$	WRONG!

In this invalid statement, I've incorrectly changed one x to an a and the other to a b. These replacements are wrong because the changes must be uniform — you must replace every x with the same constant or the same variable throughout. Here's another invalid use of **UI**:

$(Px \,\&\, Qx) \rightarrow (Rx \,\&\, Sx)$	WRONG!

In this statement, I've incorrectly tampered with the constants a and b. These changes are wrong because **UI** allows you to change only the variable that's being quantified in the original statement — in this case, x. Other variables and constants in the original statement must remain the same.

Easy rule #2: Existential generalization

REMEMBER

Existential generalization (**EG**) allows you to change either an individual constant or a free variable to a bound variable by adding brackets and quantifying it with ∃.

For example, suppose you know that this statement is true:

> My car is white.

In QL, you can express this statement as

$$Wc$$

Because you have an example of a specific thing that is white, you know more generally that *something* is white, so it's safe to say

> There exists an *x* such that *x* is white.

You can make this leap formally using **EG** like this:

$$\exists x\,[Wx]$$

Warming up with a proof

Just as **UI** allows you to break down QL statements at the beginning of a proof, **EG** allows you to build them back up at the end.

For example, suppose you want to prove that the following argument is valid:

$$\forall x\,[Px \rightarrow Qx], \forall x\,[(Pb \,\&\, Qx) \rightarrow Rx], Pa \,\&\, Pb : \exists x\,[Rx]$$

1. $\forall x\,[Px \rightarrow Qx]$	**P**
2. $\forall x\,[(Pb \,\&\, Qx) \rightarrow Rx]$	**P**
3. $Pa \,\&\, Pb$	**P**

First, use **UI** to unpack the first two premises.

4. $Pa \rightarrow Qa$	1 **UI**
5. $(Pb \,\&\, Qa) \rightarrow Ra$	2 **UI**

Line 5 illustrates a point I discuss in the previous section: You can replace both instances of the variable x in statement 2 with the constant a, but the constant b must remain unchanged.

Next, use **Simp** to break down statement 3.

6. Pa	3 **Simp**
7. Pb	3 **Simp**

Now you can call forth all your old tricks from SL proofs.

8. Qa	4,6 **MP**
9. $Pb \,\&\, Qa$	7,8 **Conj**
10. Ra	5,9 **MP**

You've now found one example of a constant (a) that has the property you're seeking (R), so you can complete the proof using **EG**.

11. $\exists x\,[Rx]$	10 **EG**

Valid and invalid uses of EG

EG gives you the same choices as **UI**, but with **EG** you're adding brackets and binding variables rather than removing brackets and freeing variables.

REMEMBER

EG allows you to change any constant to a variable and then bind that variable using the quantifier $\exists$. As with **UI**, the changes you make must be uniform throughout. For example, suppose you have this premise:

$$(Pa \,\&\, Qb) \rightarrow (Rb \vee Sa)$$

You can use **EG** to change either constant (a or b) to a variable and bind it with an $\exists$ quantifier.

$$\exists x\,[(Px \,\&\, Qb) \rightarrow (Rb \vee Sx)]$$

$$\exists x\,[(Pa \,\&\, Qx) \rightarrow (Rx \vee Sa)]$$

As with **UI**, when you use **EG**, the changes you make to constants must be uniform throughout. For example, here are three invalid uses of **EG**:

$\exists x\,[(Px \,\&\, Qx) \rightarrow (Rx \vee Sx)]$	WRONG!

In this invalid statement, I incorrectly changed both constants a and b to x.

$\exists x\,[(Px \,\&\, Qb) \rightarrow (Rb \vee Sa)]$	WRONG!

Here, I went the other direction and changed one a but not the other.

$\exists x\,[(Pa \,\&\, Qb) \rightarrow (Rb \vee Sa)]$	WRONG!

In this statement, I failed to change *any* constants to variables.

REMEMBER

EG allows you to make similar changes to statement forms. Statement forms, which include at least one free variable, won't be premises in an argument, but they may arise from using **UI** and other quantifier rules. (For more on the difference between a statement and a statement form, flip to Chapter 15.)

For example, suppose you have this statement form:

$$(Vx \,\&\, Ty) \vee (Cx \rightarrow Gy)$$

You can use **EG** to bind either variable.

$$\exists x\,[(Vx \,\&\, Ty) \vee (Cx \rightarrow Gy)]$$

$$\exists y\,[(Vx \,\&\, Ty) \vee (Cx \rightarrow Gy)]$$

You can also use **EG** to change a variable, as long as you make this change uniformly throughout.

$$\exists z\,[(Vx \,\&\, Tz) \vee (Cx \rightarrow Gz)]$$

$$\exists z\,[(Vz \,\&\, Ty) \vee (Cz \rightarrow Gy)]$$

Note that all four of these examples are still statement forms because in each case one variable is still free.

As with constants, when you use **EG** to bind or change variables, the changes need to be uniform throughout the statement form. Here are some invalid uses of **EG** applied to statement forms:

$\exists x\,[(Vx \,\&\, Tx) \vee (Cx \rightarrow Gx)]$	WRONG!

In this invalid statement, I incorrectly used **EG** to bind both variables x and y using the variable x.

$\exists z\,[(Vx \,\&\, Tz) \vee (Cx \rightarrow Gy)]$	WRONG!

Here, I changed one y to a z but failed to change the other y.

$\exists z\,[(Vx \,\&\, Ty) \vee (Cx \rightarrow Gy)]$	WRONG!

In this statement, I bound the variable z but failed to change *any* variables to z.

Not-so-easy rule #1: Existential instantiation

REMEMBER

Existential instantiation (**EI**) allows you to free the bound variable in an existential-statement (or change it to a different variable) by removing the quantifier and the brackets — *provided* this variable isn't already free in an earlier line of the proof.

You're right, this description is a lot to wrap your head around. So, to keep it simple, I explain it bit by bit. And I start by explaining everything up to the word *provided*.

How EI is similar to UI

The first part of my description of **EI** is similar to **UI**, except that it works with existential-statements (instead of universal-statements). Suppose you have the following statement:

> Something is green.

You can represent this statement in QL as follows:

$$\exists x\,[Gx]$$

As with **UI**, **EI** allows you to free the bound variable, giving you this:

Gx

This statement form means "x is green," which is close to the meaning of the original statement. **EI** also allows you to change the variable, like this:

Gy

In this case, the statement form means "y is green," which is also close to what you started with because variables are just placeholders.

EI works with variables, but not constants

Unlike **UI**, however, **EI** doesn't allow you to change the variable to a constant. For example, consider the following:

1. $\exists x\,[Gx]$	**P**
2. Ga	1 **EI** WRONG!

The reason you can't use **EI** in this way is simple: Just because the original sentence said *something* is green, you can't assume that applesauce, armadillos, Alaska, Ariana Grande, or any other *specific* thing that a may represent is also green.

REMEMBER

EI is more restrictive than **UI**. With **UI**, you start out knowing that *everything* has a given property, so you can conclude that any constant or variable also has it. But with **EI**, you start out knowing only that *something* has that property, so you can only conclude that a variable also has it.

An example using EI

To better understand how to use **EI**, consider the following argument:

$$\forall x\,[Cx \rightarrow Dx], \exists x\,[\sim Dx] : \sim \forall x\,[Cx]$$

1. $\forall x\,[Cx \rightarrow Dx]$	**P**
2. $\exists x\,[\sim Dx]$	**P**

First of all, use **UI** and **EI** to remove the quantifiers. But use **EI** *before* using **UI**. (This order has to do with stuff in the definition of **EI** that comes after the word *provided*, which I explain in the next section.)

3. $\sim Dx$	2 **EI**
4. $Cx \rightarrow Dx$	1 **UI**

The next step is obvious, even if you're not quite sure yet how using *modus tollens* (**MT**) helps. (Check out Chapter 9 for a refresher on **MT**.)

5. $\sim Cx$	3,4 **MT**

Now you know something about this variable x — namely, that x doesn't have the property C. So, you can use **EG** to make the more general statement "There exists an x such that x doesn't have the property C."

6. $\exists x\,[\sim Cx]$	5 **EG**

To complete the proof, you can use **QN**. For clarity, in this case I take an extra step and explicitly use **DN**. (See the earlier section "Introducing **QN**" for an explanation of **QN** and **DN**.)

7. $\sim \forall x \sim [\sim Cx]$	6 **QN**
8. $\sim \forall x\,[Cx]$	7 **DN**

Limits on using EI

REMEMBER

EI only allows you to free a variable that isn't free elsewhere.

In the example in the previous section, I purposely used **EI** before I used **UI** (see lines 3 and 4 in the previous section's proof). Now I explain why.

Note that the last part of the **EI** definition says, "*provided* this variable isn't already free in an earlier line of the proof." So, referring to the example in the previous section, if I had written $Cx \rightarrow Dx$ as line 3, I couldn't have used **EI** to write $\sim Dx$ in line 4.

This limitation may seem like a technicality, but trust me, it's important. To show you why, I pretend to prove an argument that's obviously false. Then I show you where the trouble comes from. Here's the argument:

Premises:

People exist.

Cats exist.

Conclusion:

Some people are cats.

The argument translated into QL looks like this:

$$\exists x\,[Px], \exists x[Cx] : \exists x\,[Px \,\&\, Cx]$$

And here's the "proof":

1. $\exists x\,[Px]$	**P**
2. $\exists x\,[Cx]$	**P**
3. Px	1 **EI**
4. Cx	2 **EI** WRONG!
5. $Px \,\&\, Cx$	3,4 **Conj**
6. $\exists x\,[Px \,\&\, Cx]$	5 **EG**

There must be a catch. Otherwise, you'd find some cat people wandering around with the rest of the human race. (I wonder if they get hairballs . . .) So, where does the problem come from?

Line 3 is fine: I use **EI** to free the variable *x*. But then in line 4, I try to use **EI** to free the variable x again. This move is a big no-no, and it leads to the mess that results. Because I've *already* freed the variable x in line 3, I'm not allowed to use this variable with **EI** in line 4.

REMEMBER

I could've written Py or Pz in line 4, but then the rest of the argument would've fallen apart. But, this is a good thing, because bad arguments are *supposed* to fall apart.

Valid and invalid uses of EI

In this section, I clarify exactly how and when you can use **EI**.

For example, suppose you're given these two premises:

1. $\exists x\,[(Rx \,\&\, Fa) \rightarrow (Hx \,\&\, Fb)]$	**P**
2. $\exists y\,[Ny]$	**P**

As with **UI**, **EI** allows you to free a variable, with the option of changing it along the way. Here are three valid next steps:

3. $(Rx \,\&\, Fa) \rightarrow (Hx \,\&\, Fb)$	1 **EI**
4. $(Ry \,\&\, Fa) \rightarrow (Hy \,\&\, Fb)$	1 **EI**
5. Nz	2 **EI**

In line 3, I used **EI** to free the variable x. This is the most common use of **EI.** In line 4, I changed the variable from x to y and then freed it using **EI**. This is also a valid option. And then in line 5, I wanted to use **EI** to free the variable from line 2. But because I had *already* freed x and y in previous lines, I needed to pick a new variable z.

As with **UI**, if you change the variable using **EI**, you must make uniform changes throughout. For example, check out this invalid statement:

6. $(Rv \,\&\, Fa) \rightarrow (Hw \,\&\, Fb)$	1 **EI** WRONG!

In line 6, I incorrectly changed one x to v and the other to w. This change is just as wrong with **EI** as it would be with **UI**.

But **EI** limits your options in ways that **UI** doesn't. For example, take a look at this invalid statement:

7. $(Ra \,\&\, Fa) \rightarrow (Ha \,\&\, Fb)$	1 **EI** WRONG!

In line 7, I uniformly changed the variable x to the individual constant a. Not allowed! With **EI**, variables must remain variables. Take a look at the last invalid statement:

8. Nx	2 **EI** WRONG!

In line 8, I changed the variable y to x. But the variable x already appears as a free variable in line 3, so you're not allowed to use it a second time with **EI**. This type of error is the most common one you're likely to make with **EI**, so watch out for it.

Not-so-easy rule #2: Universal generalization

REMEMBER

Universal generalization (**UG**) allows you to change a free variable to a bound variable by adding brackets and quantifying it with ∀ — *provided* this variable isn't free in an earlier line of the proof that's justified either by **EI** or by an undischarged assumed premise (**AP**). (Flip to Chapter 11 for more on discharging an **AP**.)

As with **EI**, **UG** is fairly straightforward up to the word *provided*. To make it as easy as possible, I explain it piece by piece.

How UG is similar to EG

For the most part, **UG** is similar to **EG**. The biggest difference is that **UG** works with universal-statements (statements with ∀) instead of existential-statements (statements with ∃), which **EG** uses. Suppose you're given the following definition:

Let Nx = x is nice

Suppose further that, at some point in a proof, you arrive at the following statement form:

Nx

Under the proper circumstances, **UG** allows you to derive the more general statement "For all x, x is nice," or, more simply put, "Everything is nice."

$\forall x\,[Nx]$

The variable you choose in this case doesn't really matter, so you can also use **UG** to write

$\forall y\,[Ny]$

UG works with variables, but not constants

Unlike **EG**, **UG** doesn't allow you to change a constant to a variable. For example, consider the following:

1. Nc	**P**
2. $\forall x\,[Nx]$	1 **UG** WRONG!

The reason this statement is wrong is simple: Just because the original sentence said that whatever c stands for (cows, coneflowers, Cadillacs, or Charli XCX) is nice, you can't jump to the conclusion that *everything* is nice.

REMEMBER

UG is more restrictive than **EG**. With **EG**, you want to show that *something* has a given property by finding one specific thing — either a constant or a variable — that has that property. With **UG**, however, you want to show that *everything* has that property by finding a variable that has it.

An example using UG

As with **EG**, **UG** is most often used at the end of a proof to build a statement form you've pieced together into a statement by quantifying it with $\forall$. Consider this argument:

$$\forall x\,[Bx \rightarrow Cx],\ \forall y\,[\sim Cy \vee Dy] :\ \forall x\,[Bx \rightarrow Dx]$$

1. $\forall x\,[Bx \rightarrow Cx]$	**P**
2. $\forall y\,[\sim Cy \vee Dy]$	**P**

The first step is to break open the two premises using **UI**.

3. $Bx \rightarrow Cx$	1 **UI**
4. $\sim Cx \vee Dx$	2 **UI**

Note that in line 4, **UI** provides the freedom to change the variable from y to x without worrying that x is already free in line 3. And once the variables are all the same, you can move forward with **Impl** and *hypothetical syllogism* (**HS**) as if you were writing an SL proof.

5. $Cx \rightarrow Dx$	4 **Impl**
6. $Bx \rightarrow Dx$	3,5 **HS**

Now, everything is set up for completing the proof with **UG**.

7. $\forall x\,[Bx \rightarrow Dx]$	6 **UG**

Limits on using UG

REMEMBER

UG only allows you to bind a variable that isn't free on a line justified by either **EI** or an undischarged **AP**.

So, here's my confession: I think **UG** is just about the orneriest concept in this whole book. And what makes **UG** a problem for me is this part of the rule about how it's *not* supposed to be used.

But look on the bright side: When you've grasped **UG**, you can rest assured that nothing worse is in store for you.

First off, imagine that you start a club that has one member: you. Even though you start off small, you have big plans for the club — eventually, you want everyone in the whole world to join. Your sense of logic is somewhat twisted, so you make this argument:

Premise:

A member of my club exists.

Conclusion:

Everyone is a member of my club.

Then, you translate this argument into QL as follows:

$\exists x\,[Mx] \therefore \forall x\,[Mx]$

This plan sounds a little fishy, but then you "prove" your argument like this:

1. $\exists x\,[Mx]$	**P**
2. Mx	1 **EI**
3. $\forall x\,[Mx]$	2 **UG** WRONG!

Somehow, in two easy steps, you increased your club membership from a single member to everything in the entire universe. Just imagine what the Taylor Swift Fan Club could do with this argument!

Here's the problem: The variable x is free on line 2, which is justified by **EI**, so you can't use **UG** to bind x in line 3. This scenario is exactly what the definition of **UG** after the word *provided* is getting at.

As another example, imagine that you're *not* a billionaire. (If this is difficult to imagine, repeat after me: "I am not a billionaire.") From this premise, you're going to try to prove that Warren Buffett isn't a billionaire. Here's your argument:

Premise:

I am not a billionaire.

Conclusion:

Warren Buffett isn't a billionaire.

The translation of this argument into QL is as follows:

$\sim Bi \therefore \sim Bw$

And now, here's the "proof":

1. $\sim Bi$	**P**
2. Bx	**AP**

Here, I use an advanced indirect proof strategy I discuss more in Chapter 12: I make up an **AP** and then work to prove a contradiction. If I'm successful, I have proved the *negation* of the **AP**. In general, this is a perfectly valid strategy, but the next step contains a fatal error.

3. $\forall x\,[Bx]$	1 **UG** WRONG!
4. Bi	2 **UI**
5. $Bi\ \&\ \sim Bi$	1,4 **Conj**
6. $\sim Bx$	2–5 **IP**

Having discharged my **AP**, the rest of the proof follows like this:

7. $\forall x\,[\sim Bx]$	6 **UG**
8. $\sim Bw$	7 **UI**

The problem this time is that the variable x is free in line 2, which is justified by an *undischarged* **AP**, which means you can't use **UG** to bind x in line 3.

Notice, however, that in line 7, the proof includes a perfectly valid usage of **UG** to bind x. The only difference is that by this point in the proof, the **AP** has been discharged.

REMEMBER

The moral of the story is that whenever you want to use **UG**, you need to check first to make sure the variable you're trying to bind isn't free in either of the following:

- A line that's justified by **EI**
- A line that's justified by an **AP** that has *not* been discharged

Valid and invalid uses of UG

Here's a summary of when you can and can't use **UG**. Suppose you have the following proof in progress (this takes a little setting up):

1. $\forall x\,[Tx]$	**P**
2. $\exists y\,[Gy]$	**P**
3. Sa	**P**
4. Tx	1 **UI**
5. Gy	2 **EI**
6. $Tx\ \&\ Sa$	3,4 **Conj**
7. $(Tx\ \&\ Sa)\ \&\ Gy$	5,6 **Conj**
8. Hz	**AP**

Now, you can use **UG** to bind the variable x in line 7.

9. $\forall x\,[(Tx\ \&\ Sa)\ \&\ Gy]$	7 **UG**

But you *can't* use **UG** to change the constant a in line 7 to a variable and bind it, because **UG** doesn't work with constants.

10. $\forall w\,[(Tx \,\&\, Sw) \,\&\, Gy]$	7 **UG** WRONG!

You also *can't* use **UG** to bind the variable y in line 7, because this variable appears free on line 5, which is justified by **EI**.

11. $\forall y\,[(Tx \,\&\, Sa) \,\&\, Gy]$	7 **UG** WRONG!

Furthermore, you *can't* use **UG** to bind the variable z in line 8, because this variable is justified by an **AP** that is still undischarged.

12. $\forall z\,[Hz]$	8 **UG** WRONG!

TIP

This example encapsulates the lion's share of the fine-print exclusions that come along with **UG**. As I mention at the beginning of the previous section, I think these **UG** rules are the toughest thing to understand in this whole book. If you can hang on to them long enough to use them reasonably well on your final logic exam, I believe you'll be well ahead of the curve when your final grades are posted.

IN THIS CHAPTER

- Understanding QL relational expressions
- Unmasking identities
- Writing proofs using relations and identities

Chapter 18
Expanding Logical Relations

In sentential logic (SL), you represent a basic statement with just one letter — a constant. For example, you can translate the statement "Barb is a tree surgeon" as

B (SL translation)

In quantification logic (QL), you represent the same sentence with a two-letter combination of a property constant and an individual constant. For example:

Tb (QL translation)

This representation works great. But suppose you want to translate the statement "Barb hired Marty" into QL. Now that more than one person is in the picture, you need a way to express their relationship.

Or suppose you want to translate the statement "Barb is the best tree surgeon in the county." In this case, you aren't describing a property that applies to Barb, but rather her identity as the *single* best tree surgeon in the county.

Fortunately, QL allows you to express these more sophisticated ideas easily. In this chapter, I discuss both relations and identities in QL. *Relations* allow you to express statements that have more than one key player, and *identities* help you handle situations where the key player is identified with an alternative but uniquely fitting description. Finally, I show you how these new concepts fit into the big picture of QL.

Relating to QL Relations

In previous chapters, you use *monadic expressions:* statements and statement forms that have only one individual constant or variable. For example, the statement

Na

has only one individual constant: a. Similarly, the statement form

Nx

has only one variable: x.

In this section, I help you expand your view of expressions to include those that have more than one individual constant or variable.

Defining and using relations

REMEMBER

A *relational expression* has more than one individual constant or variable.

Defining a relational expression is quite easy after you know the basics of defining a monadic expression (see Chapter 15). For example, you can define a monadic expression as follows:

Let $Nx = x$ is nosy

Then, you may translate the sentence "Arkady is nosy" as

Na

But suppose you have the statement

Arkady is nosier than Boris.

In this case, you can begin by defining the following relational expression:

Let Nxy = x is nosier than y

Then, you can use this expression to translate the statement as follows:

Nab

Similarly, you can translate the statement

Boris is nosier than Arkady

as

Nba

WARNING

The order of the individual constants or variables in a relational expression is crucial — don't mix them up!

Connecting relational expressions

You can use the five SL operators from Chapter 4 (~, &, ∨, →, and ↔) to connect relational statements and statement forms just as you use them with monadic expressions. (For more on distinguishing statements from statement forms, flip to Chapter 15.)

REMEMBER

After you translate a statement from English into a relational expression, it becomes an indivisible unit just like a monadic expression. From then on, the same rules for using the five operators apply.

For example, suppose you want to say

Kate is taller than Christopher, but Christopher is taller than Paula.

First, define the constants.

Let k = Kate

Let c = Christopher

Let p = Paula

Next, define the appropriate relational expression.

Let Txy = x is taller than y

Now, you can translate the statement to

$Tkc \,\&\, Tcp$

Similarly, you can translate the statement

If Kate is taller than Christopher, then she is also taller than Paula.

as

$Tkc \rightarrow Tkp$

Making use of quantifiers with relations

You can use the two QL quantifiers from Chapter 15 ($\forall$ and $\exists$) to connect relational statements and statement forms just as you do with monadic expressions.

REMEMBER

Using quantifiers with relations isn't substantially different from using them with monadic expressions.

For example, suppose you want to translate the following English statement into QL:

Everybody likes chocolate.

First define the relational expression.

Let Lxy = x likes y

Then, define the individual constant.

Let c = chocolate

Now, you're ready to translate the statement as follows:

$\forall x\,[Lxc]$

Similarly, suppose you have the following expression:

Someone introduced Dolly to Jack.

This time, you need to define a relational expression that takes three individual variables.

Let $Ixyz$ = x introduced y to z

Using the initials for Dolly and Jack as individual constants, and binding the variable x with the quantifier $\exists$, you can translate the statement as

$$\exists x\,[Ixdj]$$

You can also translate the statement

> Jack introduced Dolly to everyone.

as

$$\forall x\,[Ijdx]$$

TECHNICAL STUFF

Notice that you can still use the variable x here even though the definition for the relational expression uses z in this position. The choice of variable isn't important as long you've quantified it properly.

Working with multiple quantifiers

Because relations have more than one variable, they open up the possibility of statements with more than one quantifier. For example, you can translate the statement

> Everybody introduced someone to Jack.

as

$$\forall x\ \exists y\,[Ixyj]$$

By breaking down this QL statement, you realize that it literally says: "For all *x*, there exists a *y* such that *x* introduced *y* to Jack."

Similarly, if you want to translate the following statement into QL:

> Jack introduced someone to everyone.

You can use the following statement:

$$\forall x\ \exists y\,[Ijyx]$$

This statement means "For all *x*, there exists a *y* such that Jack introduced *y* to *x*."

WARNING

Be careful with the order of quantifiers. Changing the order from $\forall x \exists y$ to $\exists y \forall x$ changes the meaning of a statement even if the contents of the brackets remain the same.

Here's the distinction between statements starting with $\forall x \exists y$ and those starting with $\exists y \forall x$:

REMEMBER

- Generally speaking, $\forall x \exists y$ means "For all x, there exists a y such that . . .", which means that *y may be different* for two different values of x.
- On the other hand, $\exists y \forall x$ means "There exists a y such that for all x . . .", which means that *y is the same* for every x.

To make this distinction clear, I define a new property constant.

Let Mxy = x is married to y

Now, suppose I want to express the idea that all people are married. I can translate the statement as

$$\forall x \; \exists y \; [Mxy]$$

This statement literally means "For all x, there exists a y such that x is married to y."

But suppose I reverse the quantifiers, like this:

$$\exists y \; \forall x \; [Mxy]$$

Now it means "There exists a y such that for all x, x is married to y." In this case, I'm saying that all people are married to *the same person*. This is definitely not what I had in mind.

TIP

Multiple quantifiers are especially useful for expressing mathematical ideas. For example, suppose you want to express the idea that the counting numbers (1, 2, 3, . . .) are infinite. You can express this idea by defining the following relational expression:

Let Gxy = x is greater than y

Now you can express the infinity of all numbers by stating that for every number, there exists a number that's greater.

$$\forall x \; \exists y \; [Gyx]$$

In other words, "For all x, there exists a y such that y is greater than x."

Writing proofs with relations

REMEMBER

When you're writing proofs, relational expressions work pretty much the same as monadic expressions. They stand as discrete and inseparable chunks that you can manipulate using the 18 rules of inference for SL (see Chapters 9 and 10).

Quantifier negation (**QN**) and the four quantifier rules — *universal instantiation* (**UI**), *existential generalization* (**EG**), *existential instantiation* (**EI**), and *universal generalization* (**UG**) — also work the same with relational expressions as they do with monadic expressions. (I cover these fabulous QL rules in Chapter 17.) The main difference is, now you may need to write proofs that include statements with multiple quantifiers.

Using QN with multiple quantifiers

QN works the same with multiple quantifiers as it does with single quantifiers. Just make sure you're clear which quantifier you're changing.

For example, here's a new declaration:

> Let Pxy = x gave a present to y

Suppose you want to prove the following argument:

Premise:

> Everyone gave at least one person a present.

Conclusion:

> Nobody didn't give a present to anybody.

You can write this argument in QL as follows:

$$\forall x\, \exists y\, [Pxy] : \sim \exists x\, \forall y\, \sim [Pxy]$$

1. $\forall x\, \exists y\, [Pxy]$	**P**
2. $\sim \exists x \sim \exists y\, [Pxy]$	1 **QN**

In line 2, applying **QN** to the first quantifier, I placed a not-operator before this quantifier, changed it from $\forall x$ to $\exists x$, and placed another not-operator after this new quantifier.

3. $\sim\exists x \sim\sim \forall y \sim[Pxy]$	2 **QN**
4. $\sim\exists x\ \forall y \sim[Pxy]$	3 **DN**

In line 3, the second quantifier is now the focus of the change from line 2. Again applying **QN**, I've added not-operators before and after this quantifier and changed it from $\exists y$ to $\forall y$. To keep the proof crystal clear, I've written an additional line to apply the *double negation* (**DN**) rule explicitly (see Chapter 10).

Applying the four quantifier rules with multiple quantifiers

REMEMBER

When you're using the four quantifier rules — **UI**, **EG**, **EI**, and **UG** (see Chapter 17) — with multiple quantifiers, an important restriction comes into play: You can only remove the first (leftmost) quantifier or add a new first quantifier.

In practice, this restriction means you have to do the following:

- Break statements down from the outside in.
- Build statements up from the inside out.

For example, take a look at this proof:

$$\forall x\ \forall y\ [Jxy \rightarrow Kyx],\ \exists x \sim\exists y\ [Kyx] :\ \exists x\ \forall y\ [\sim Jxy]$$

1. $\forall x\ \forall y\ [Jxy \rightarrow Kyx]$	**P**
2. $\exists x \sim\exists y\ [Kyx]$	**P**

First, do a quick switch of quantifiers in the second premise to move the not-operator to the right and get it out of the way.

3. $\exists x\ \forall y \sim[Kyx]$	2 **QN**

But this premise still has an ∃-quantifier, so you have to use **EI**. And you need to get this step out of the way as soon as possible before *x* shows up free in the proof (see Chapter 17 for more on how **EI** works). So, now you go about breaking it down from the outside in.

4. $\forall y \sim [Kyx]$	3 **EI**
5. $\sim Kyx$	4 **UI**

Now you can tackle the first premise, again breaking it down from the outside in.

6. $\forall y [Jxy \rightarrow Kyx]$	1 **UI**
7. $Jxy \rightarrow Kyx$	6 **UI**

The next step using *modus tollens* (**MT**) from Chapter 9 is pretty clear.

8. $\sim Jxy$	5,7 **MT**

Then it's time to build the conclusion from the inside out.

9. $\forall y [\sim Jxy]$	8 **UG**

REGARDING SELF-REFERENTIAL STATEMENTS

A few fine points to proofs with relations are beyond the scope of this book. Many of these issues arise when you're working with *self-referential statements*, which are relational expressions with a repeated individual constant.

For example, when given the statement "Jack loves everybody" — $\forall x [Ljx]$ — you can use **UI** to infer the statement "Jack loves himself" — Ljj.

Self-referential statements are necessary to make QL fully expressive, but you need to keep a close eye on them in proofs. As I mention in Chapter 22, self-referential statements open the door to paradoxes such as Russell's paradox, which can introduce inconsistencies into logical systems unless they're carefully regulated.

This use of **UG** is valid because in line 4, the variable y is still bound.

Finally, you use **EG** to finish off the proof.

10. $\exists x\ \forall x\ [\sim Jxy]$	9 **EG**

Identifying Constants with Identities

Look at the following two statements:

> George Washington was a U.S. president.
>
> George Washington was the first U.S. president.

You can translate the first statement into QL easily as

Pg

At first glance, you may think you can handle the second statement with the same approach. But you can't, because these two statements are quite different, even though they look similar.

The first statement describes Washington in terms of a *property* that's possibly shared by others (for example, by Abraham Lincoln). That's why you can easily use a *property constant* to translate the statement into QL (see Chapter 15 for more on property constants).

The second statement, however, tells you about Washington in terms of his *identity* as the one and only first U.S. president. To translate this statement into QL, you need something new — and this something new is a way to express identity. I discuss the ins and outs of identities in the following sections.

Understanding identities

REMEMBER

An *identity* tells you that two different individual constants refer to the same thing, meaning they're interchangeable in QL.

What does it mean to say that George Washington was *the* first U.S. president? Essentially, you're saying that anytime you talk about *George Washington*, you can substitute the words *the first U.S. president*, and vice versa.

In QL, you can state an identity formally as follows (assuming g stands for *George Washington* and f stands for *the first U.S. president*):

$$g = f$$

You can also use an identity with a quantifier, like this:

$$\exists x\,[x = f]$$

This statement translates to "There exists an x such that x is the first U.S. president" or, more simply, "The first U.S. president exists."

IDENTITY CRISES

You should be aware of some intriguing exceptions to the idea that *identity* (see the section "Understanding identities" also in this chapter) means you can substitute individual constants freely from one statement to another without changing the meaning. For example, look at these two statements:

Sally *believed* that Abraham Lincoln was the first U.S. president.

Sally *believed* that Abraham Lincoln was George Washington.

Clearly, these statements don't mean the same thing, and the first statement can be true and the second false.

For this reason, QL doesn't work with statements that contain certain types of sub-statements, such as

Elton *hoped* that Mary had paid the gas bill.

The boss *insisted* that everybody arrive on time.

We *know* that Clarissa is lying.

It is *necessary* that the sun rise every morning.

See Chapter 21 for a discussion of non-classical systems of logic that attempt to account for statements of these kinds.

But remember that, as far as QL is concerned, once you define an identity between two constants, you're saying it's okay to substitute one constant for the other.

Writing proofs with identities

QL contains two rules just for handling identities. They're so easy to understand and use that I give you just one example of each.

Rule of identity

The *rule of identity* **(ID)** simply provides formally for the substitution of one constant for another in a proof after the identity of these constants has been shown.

REMEMBER

After you establish an identity of the form $x = y$ in a proof, **ID** allows you to rewrite any line of that proof by substituting x for y (or y for x).

Here's an argument that requires an identity:

Premises:

Every musician deserves respect.

Chappell Roan is a musician.

Chappell Roan is the 2025 Grammy Award winner for Best New Artist.

Conclusion:

The 2025 Grammy Award winner for Best New Artist deserves respect.

Here's the same argument translated into QL:

$$\forall x\,[Mx \rightarrow Dx],\ Mc,\ c = g :\ Dg$$

The proof of this argument is straightforward.

1. $\forall x\,[Mx \rightarrow Dx]$	**P**
2. Mc	**P**
3. $c = g$	**P**
4. $Mc \rightarrow Dc$	1 **UI**
5. Dc	2,4 **MP**
6. Dg	5 **ID**

I used **ID** on the last step to substitute g for c. In most cases, when you need to use **ID**, you'll be able to see the opportunity from miles away.

Identity reflexivity

Identity reflexivity (**IR**) is even easier to understand than **ID**. **IR** just tells you that assuming everything is identical to itself is okay — no matter what you're proving.

REMEMBER

IR allows you to insert the statement $\forall x\,[x = x]$ at any point in a proof.

The statement $\forall x\,[x = x]$ translates into English as "For all x, x is identical to x" or, more simply, "Everything is identical to itself."

IR is one of those rules you'll almost never need unless your teacher sets up a proof on an exam that requires it (which is always possible). For example, take a look at the following proof:

$$\forall x\,[((x = m) \vee (x = n)) \rightarrow Tx] :\ Tm \,\&\, Tn$$

1. $\forall x\,[((x = m) \vee (x = n)) \rightarrow Tx]$	**P**
2. $((m = m) \vee (m = n)) \rightarrow Tm$	1 **UI**
3. $((n = m) \vee (n = n)) \rightarrow Tn$	1 **UI**

I used **UI** to unpack the premise in two different ways: In line 2, changing the variable x to the constant m, and in line 3, changing x to n. Now, you can pull out **IR**.

4. $\forall x\,[x = x]$	**IR**

TIP

IR is like an extra premise, so you don't need to reference any line numbers when you use it. After using **IR**, you can use **UI** to get the identity statements you need.

5. $m = m$	4 **UI**
6. $n = n$	4 **UI**

This time, I used **UI** to unpack line 4 in two different ways, again changing x first to m and then to n.

Now everything is in place to finish the proof using only rules of inference from SL (see Chapter 9): *addition* (**Add**), *modus ponens* (**MP**), and *conjunction* (**Conj**).

7. $(m = m) \vee (m = n)$	5 **Add**
8. $(n = m) \vee (n = n)$	6 **Add**
9. Tm	2,7 **MP**
10. Tn	3,8 **MP**
11. $Tm \,\&\, Tn$	9,10 **Conj**

IN THIS CHAPTER

- Extending the truth tree method to QL statements
- Pondering non-terminating truth trees

Chapter 19
Planting a Quantity of Truth Trees

In Chapter 8, I show you how to use truth trees in sentential logic (SL) for a variety of different purposes. In this chapter, you see how you can extend this method to quantification logic (QL). As in SL, truth trees in QL are generally simpler than proofs. You don't need to have a brainstorm to make them work — you just plug and chug.

So, in this chapter, I show you how to make the most of the truth tree method for solving problems in QL. However, I warn you right up front that (unfortunately) QL truth trees have limitations. When you get comfortable with using truth trees in QL, I reveal an important drawback: the non-terminating tree.

Applying Your Truth Tree Knowledge to QL

Everything you know about building truth trees in SL also applies to QL trees. In this section, I give you a QL example to show you how it's done. If at any time you come across something unfamiliar and get stuck, flip to Chapter 8 for a refresher.

Using the decomposition rules from SL

Suppose you want to test whether the following set of three statements is consistent:

$Ma \vee \sim Tb$

$\sim Ma \,\&\, Lc$

$\sim Tb$

As with truth trees in SL, the first step in deciding whether statements are consistent or inconsistent is to build the trunk of the tree using these three statements:

$Ma \vee \sim Tb$

$\sim Ma \,\&\, Lc$

$\textcircled{\sim Tb}$

REMEMBER

Notice that I circled the third statement because it's already in a form that can't be decomposed further.

Now, I can build the tree using the decomposition rules from SL. I start with the second statement because it's a single-branching statement.

$Ma \vee \sim Tb$

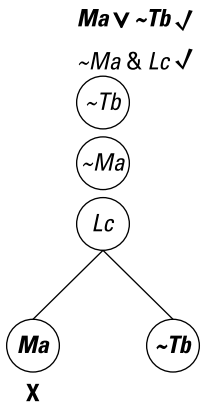
$\mathbf{\sim Ma \,\&\, Lc \checkmark}$

$\textcircled{\sim Tb}$

$\textcircled{\mathbf{\sim Ma}}$

$\textcircled{\mathbf{Lc}}$

Having decomposed $\sim Ma \,\&\, Lc$, I checked this statement to show that I'm done with it. And again, I circled the resulting statements that are fully decomposed. The only statement left that isn't checked or circled is the first statement.



Now, every statement is either checked or circled, so the tree is finished.

Notice that I also closed off the branch that ends with *Ma* by placing an X below it. As with SL trees, the reason for closing off this branch is simple: To travel all the way from the beginning of the trunk to the end of this branch, you need to pass through both $\sim Ma$ and Ma. Because a statement and its negation can't both be true, the branch is closed off. And this is the only reason to close off a branch.

But the branch that ends with $\sim Tb$ has no such contradiction, so I left it open. As with SL trees, because this truth tree has at least one open branch, the set of three statements is considered *consistent*.

Adding UI, EI, and QN

When you're working with statements with quantifiers, you need to rely on some of the QL quantifier rules from Chapter 17. For breaking down statements in truth trees, use *universal instantiation* (**UI**) and *existential instantiation* (**EI**). And for removing the not-operator from a quantifier, use *quantifier negation* (**QN**).

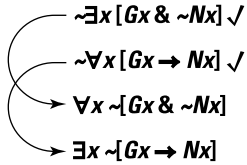
Here's an argument with one premise, which I test for validity using a truth tree:

$$\sim\exists x\,[Gx\ \&\ \sim Nx]\,:\ \forall x\,[Gx \rightarrow Nx]$$

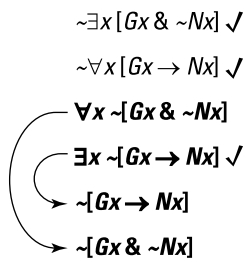
As always, the first step is to build the trunk of the tree by using the premise and the *negation* of the conclusion.

$$\sim\exists x\,[Gx\ \&\ \sim Nx]$$
$$\sim\forall x\,[Gx \rightarrow Nx]$$

Because both statements have negated quantifiers, use **QN** to move the not-operators. (Note that you don't have to use line justifications such as **QN**, **UI**, or **EI** with a truth tree as you do with a proof.) I save space here by taking two steps at once.



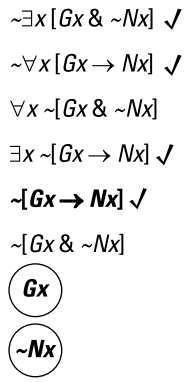
Now you're ready to use **UI** and **EI**. But remember the limitation on **EI**: Just as in QL proofs, you can only use **EI** to free a variable that isn't already free (see Chapter 15 for a description of free variables). So, in this example, you have to get **EI** out of the way before using **UI**.



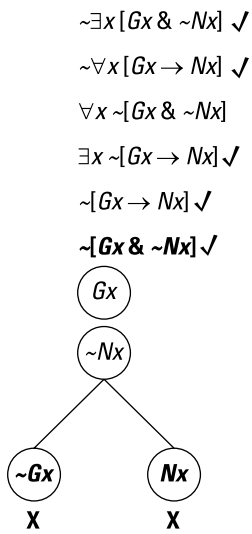
WARNING

When you use **UI**, don't check the statement that you just decomposed. I tell you why in the next section, "Using UI more than once." But, for now, just remember this rule: When you use **EI**, check the decomposed statement as usual, but don't check it when you use **UI**.

At this point, you can start decomposing statements using the rules from SL truth trees. When possible, start with a single-branching statement.



Finally, decompose the double-branching statement.



Both branches lead to contradictions, so you can close off both with Xs. So even though one statement is still not checked off, because every branch has been closed off, the tree is finished and the argument is valid.

Applying UI more than once

In the example from the previous section, I tell you that when you use **UI** to decompose a statement, you do *not* check it as complete. I promised an explanation, and here it comes.

REMEMBER

When you decompose a universal-statement using **UI**, you have an unlimited number of constants you can choose for the decomposition. So, leave the statement unchecked because you may need to use it again.

For example, consider this argument :

$\forall x\,[Hx \rightarrow Jx],\ Ha \,\&\, Hb :\ Ja \,\&\, Jb$

I set up the trunk of the tree as usual, using the premises and the negation of the conclusion.

$\forall x\,[Hx \rightarrow Jx]$
$Ha \,\&\, Hb$
$\sim(Ja \,\&\, Jb)$

I know I need to decompose the first premise using **UI**, but I have many choices. I can decompose it to $Hx \rightarrow Jx$, or $Ha \rightarrow Ja$, or $Hb \rightarrow Jb$, or an infinite number of other possible statements. As this example shows, I may need more than one of these decompositions to complete the tree, so I have to leave the universal-statement unchecked so I'm free to decompose it again as needed.

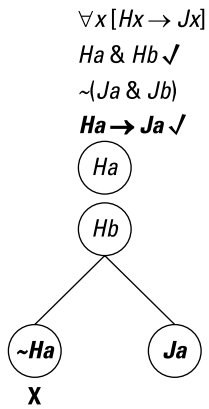
For my first decomposition, I want to go in a direction that looks useful, so I choose a decomposition that includes one of the constants that appears in other statements in the tree. I start with the constant a.

$\boldsymbol{\forall x\,[Hx \rightarrow Jx]}$
$Ha \,\&\, Hb$
$\sim(Ja \,\&\, Jb)$
$\boldsymbol{Ha \rightarrow Ja}$

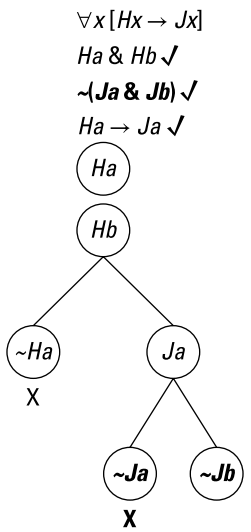
I leave the statement that I just decomposed unchecked, and move on to the next step.

$\forall x\,[Hx \rightarrow Jx]$
$\boldsymbol{Ha \,\&\, Hb}$ ✓
$\sim(Ja \,\&\, Jb)$
$Ha \rightarrow Ja$
$\boldsymbol{Ha}$
$\boldsymbol{Hb}$

This time, I checked the decomposed statement as usual. Now I have to double-branch.



Fortunately, one of the branches gets closed off, and I can start on the next step.



WARNING

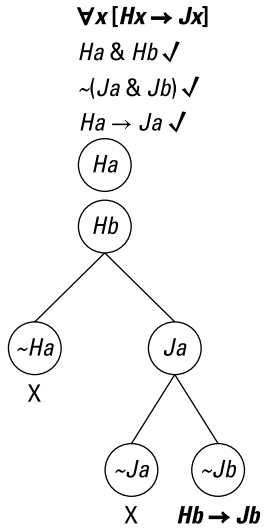
You may be tempted at this point to think the tree is finished. But remember, the tree is only finished when either of the following is true:

- Every statement has been either checked or circled.
- Every branch has been closed off.

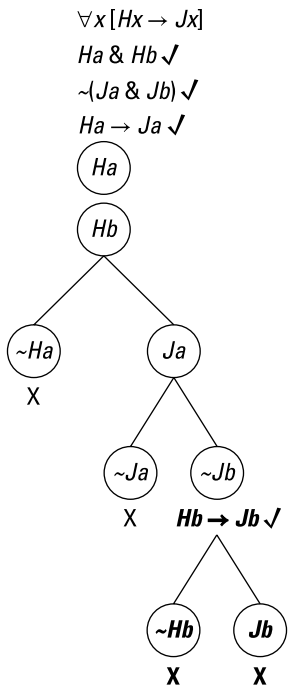
If these rules don't sound familiar, check out Chapter 8.

Here's the key point: The reason the tree isn't finished yet is because when I used **UI** to decompose the first statement, I didn't check it off as complete. So, now, I decompose this statement in a different way that allows me to complete the tree.

In this case, I decompose the unchecked universal-statement using the other constant that appears in the tree — namely, the constant b.



Note that even when I use **UI** to decompose the first statement for the second (or third, or hundredth) time, I leave the statement unchecked in case I need to use it again. Now, just one final decomposition and I'm done.



With this last decomposition, every branch is closed off, so, even though one statement is still left unchecked, the tree is complete. Because all the branches are closed off, you know the argument is valid.

Reaching for the Sky: Non-Terminating Trees

SL truth trees are my favorite logical tool because they don't require you to be clever. You just have to follow the right steps to the end, and you get the correct answer every time.

REMEMBER

Unfortunately, QL truth trees tend to be a bit more unruly. In some cases, a tree just grows and grows and grows — and never stops growing. This type of tree is called a *non-terminating tree* (or an *infinite tree*), and it makes things interesting, if not somewhat frustrating.

As I say in the previous section, you have only two ways to finish a tree: Either check or circle every statement *or* close off every branch with an X. But, in QL, it just isn't always possible to complete a truth tree.

To illustrate, here's a statement that I translate into QL and then test using a truth tree:

> The counting numbers aren't infinite.

This is a *false* statement, because the counting numbers (1, 2, 3, . . .) go on forever, so they *are* infinite. But it's a statement you can express in QL. First, I declare a domain of discourse.

> Domain: Counting numbers

Now, I define a relation.

> Lxy = x is less than y

I want to express the idea that no matter which number x I choose, that number is less than some other number y, and then negate it. Here's how I write it:

> $\sim \forall x \exists y\, [Lxy]$

This translates literally as "It isn't true that for all x, there exists a y such that x is less than y" or, more informally, "The counting numbers aren't infinite."

Suppose I want to test if this statement is a *tautology* (a statement that's always true; see Chapter 6). It better *not* be a tautology, because I already have an interpretation that makes it false.

As I point out in Chapter 8, to use a truth tree to see whether a statement is a tautology, negate that statement and use it as your trunk. When you complete the tree, if at least one open branch remains, the statement is a tautology; otherwise, it's either a *contradiction* (a statement that's always false) or a *contingent statement* (which can be true or false depending on the interpretation).

So, my first step is to negate the statement and use it as the trunk of my tree. Then I decompose it using **UI** and remembering *not* to check off the first statement. To save space, I show both steps at once.

$$\forall x \exists y\, [Lxy]$$

$$\exists y\, [Lxy]$$

Next, I decompose this new statement using **EI**:

$$\forall x \exists y [Lxy]$$
$$\mathbf{\exists y [Lxy]} \checkmark$$
$$\mathbf{(Lxz)}$$

In this case, I decomposed the statement by changing the variable to z in the hopes that I may be able to get rid of this variable later by applying **UI** to the first statement again.

At this point, the tree isn't complete because a branch is open and a statement isn't checked. But, I still have a chance to close off the branch with a clever use of **UI** applied to the first statement. And, because the variable z is now in the mix, I use this constant in my decomposition.

$$\mathbf{\forall x \exists y [Lxy]}$$
$$\exists y [Lxy] \checkmark$$
$$(Lxz)$$
$$\mathbf{\exists y [Lzy]}$$

Now, it's time for another application of **EI**.

$$\forall x \exists y [Lxy]$$
$$\exists y [Lxy] \checkmark$$
$$(Lxz)$$
$$\exists y [Lzy] \checkmark$$
$$\mathbf{(Lzw)}$$

This time, I introduce the variable w, but it doesn't matter. After two iterations, you can probably see that nothing you do can complete the tree, either by closing off the branch or checking off the first statement: The universal-statement can be decomposed to infinitely many statements, but never checked off. What you have here is a non-terminating tree.

Because the tree is never finished, it tells you nothing about whether the statement you're testing is a tautology. Non-terminating trees are the reason that truth trees, which are always useful in SL, have a more limited function in QL.

5 Meditating on Modern Developments in Logic

IN THIS PART . . .

Recognize how logic is instrumental to computer hardware and software.

Play with a few examples of nonclassical logic.

Examine how paradoxes challenge logic, and how questions of consistency and completeness in logic led to the mathematical discovery of the century.

IN THIS CHAPTER

- Appreciating the earliest computers
- Understanding how logic works in today's computers
- Addressing the rise of AI

Chapter 20
Computer Logic

The computer has been called the most important invention of the 20th century (well, except for maybe the auto-drip coffee maker). And what sets the computer apart from other inventions — for example, the airplane, the radio, the television, or the nuclear power generator — is its versatility.

When you think about it, most machines are just tools for doing repetitive work that humans don't like to do. And honestly, machines generally do their jobs better than humans could ever do them. From the can opener to the car wash, machines have long been built to mimic human movement and then improve upon it.

So, it makes sense that along the way, people began to wonder whether it was possible to build a machine to take over some of the repetitive *mental* labor humans must do day in and day out. To some extent, adding machines and cash registers were invented to do just that. But these inventions are also limited to their intended functions. Just as you can't expect a can opener to wash a car, you can't expect an adding machine to do long division, let alone differential calculus. A few visionaries, however, saw the possibility that a single machine might be capable of performing an unlimited number of logical functions.

In this chapter, I explain the role logic played in the design of the computer. I start out with the beginnings of the computer, recognizing the work of Charles Babbage and Ada Lovelace. Then, I discuss how Alan Turing showed, in theory at least, that a computer can perform any calculation a human can do. Next, I focus on the ways in which logic forms the underpinnings of computer hardware and software. Finally, I examine how computers have evolved in the age of artificial intelligence.

Remembering the Early Versions of Computers

Even though the work of building the first electronic computers began in the 1940s, the idea and design for them took root more than a century earlier. The computer began as a crackpot idea that never really got off the ground, and it developed into one of the most important inventions in history.

Babbage designs the first computers

Charles Babbage (1791–1871) is credited as the inventor of the computer. Even though his two models — the *difference engine* and the later *analytical engine* — were both designed to be powered by mechanical energy rather than electrical energy, they were highly sophisticated machines and had much more in common with later computers than with other inventions of the day.

Babbage began working on the difference engine in the 1820s. Unfortunately, even though he completed its design, he never finished building the machine. Funding difficulties and Babbage's personality conflicts with others on the project are cited as the reasons for the failure to finish. (I'm sure modern-day computer engineers can relate to these types of obstacles.)

It wasn't until 1991 that the first and only difference engine was built in accordance with Babbage's plans. The builders limited themselves to the technology that would've been available in Babbage's time. They found that the machine worked in accordance with his plans, performing complex mathematical calculations with perfect accuracy.

After Babbage abandoned his plan to build the difference engine, he took up a more ambitious project — designing the analytical engine. This project incorporated the skills he developed from designing the difference engine, but he took them a step further. One major improvement: The analytical engine could be programmed with punch cards, making it more versatile and easier to use than the difference engine.

Ada Lovelace (1815–1852), a mathematician and friend of Babbage, assisted greatly in the design of the analytical engine. She also wrote several programs that would have run on the engine had it been built.

Turing and his universal Turing machine

After Charles Babbage's death in 1871, his designs gathered dust for decades until another visionary — Alan Turing (1912–1954) — approached the idea of mechanized computing from a different angle.

Turing saw the need to clarify exactly what was meant by computation, which humans had been doing for centuries by using *algorithms*. Algorithms are simply mechanical procedures that produce a desired result in a finite number of steps. For example, the procedure for multiplying two numbers is an algorithm. As long as you follow the steps correctly, you're guaranteed to eventually get the right answer, no matter how big the numbers are.

REMEMBER

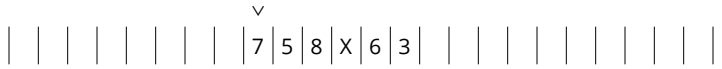
Turing found he was able to break down algorithms into small enough steps that an extremely simple machine could perform them. He called this computing machine the *universal Turing machine* (*UTM*).

Unlike Babbage, Turing never expected his machine to be built. Instead, it was a theoretical model that was described in principle without needing to be realized. However, the basic capability of a UTM is shared by all classical (that is, nonquantum) computers. In other words, every classical computer, no matter its design, is neither more nor less capable of calculation than any other. (For a look at how classical and quantum computers differ, see Chapter 21.)

Explaining the UTM

The UTM consists of a paper strip of random length that's divided into boxes. Each box contains a single symbol from a finite set of symbols. The paper strip is on rollers that move it one box at a time past a pointer, which can both read the symbol in each square and, in some cases, erase it and write a different symbol.

For example, suppose you want to write a program to multiply two numbers together. You'd begin by writing two numbers on the strip, separating them with the multiplication operator:



These numbers — 758 and 63 — are the program's *initial conditions.* By the end of the algorithmic procedure, the result will be the number 47,754, which looks like this:

REMEMBER

Turing laid out a set of allowable steps for getting from start to finish. These steps form what's called a *program*. A program consists of a list of *states* that tell the machine which *actions* to take based on what the pointer reads. Depending upon which state the machine is in, it performs different specific actions. The general form of these actions are as follows:

1. Optionally change the symbol in the box to a different symbol.
2. Optionally change the state that the machine is in.
3. Move one box to the right or left.

For example, when the machine is in state #1, and the pointer reads the number 7, the actions may be

1. Leave the 7 unchanged.
2. Change the machine from state #1 to state #10.
3. Move one box to the left.

However, when the machine is in state #2, and the pointer reads the number 7, the actions may be

1. Change the 7 to an 8.
2. Leave the state unchanged.
3. Move one box to the right.

Even though these actions seem crude, Turing showed that it's possible to perform sophisticated calculations in this way. Far more important, he proved that it's possible to perform *any* calculation in this way. That is, any algorithmic method a human can learn to do, the UTM can also be programmed to do.

Relating the UTM to logic

But you may ask, what does the UTM have to do with logic? To understand the connection, notice that the specifications for the machine make deciding the truth

value of certain types of statements simple. For example, at the start of the multiplication problem in the previous section, this statement is true:

> The machine is in state #1, and the pointer is on 7.

Based on the truth of this statement, the machine performs the proper actions, which resets the machine so that now this statement is true:

> The machine is in state #10, and the pointer is on 5.

REMEMBER

Logic is ideal for describing the conditions of the UTM at any time. As you discover in the next section, computer scientists have capitalized on the descriptive power of logic in the design of both hardware and software.

Decoding the Modern Age of Computers

The early ideas and theories of Babbage and Turing — computer development pioneers — set the stage for the evolution of the modern computer. And, with electric power becoming more and more commonplace, the next advance in computers soon followed. Built in the 1940s, ENIAC (short for Electronic Numerical Integrator and Computer) was the first electronic computer. Improved models developed swiftly.

All computers people have engineered (so far) — including your home computer or laptop — can be divided into two levels of functionality:

- **Hardware:** The physical structure of the computer
- **Software:** The programs that run on the computer

As I explain in the following sections, logic plays an integral role in computing at both the hardware and software levels.

Getting the basics of hardware and logic gates

In logic, a *truth function* takes one or two input values and then generates an output value. Input and output values can be either of the two truth values (**T** or **F**). (See Chapter 13 for a refresher on truth functions.)

All five logical operators in sentential logic (SL) are truth functions. Three of them — the not-operator, the and-operator, and the or-operator — are sufficient to build every possible truth function. Similarly, the (not-both) nand–operator (represented by the symbol | and also known as the *Sheffer stroke*) by itself is sufficient for building every possible truth function. (See Chapter 13 for an expanded understanding of these ideas.)

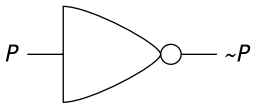
At the basic level, computer circuits mimic truth functions. Instead of using the values **T** and **F**, however, computers use the values 1 and 0 as in Boolean algebra (see Chapter 14). In honor of George Boole, variables taking only these two values are called *Boolean variables*. When the current flow in a specific part of the circuit is on, the value there is said to be 1; when it's off, the value there is 0.

Types of logic gates

REMEMBER

The behavior of logical operators is mimicked by *logic gates*, which allow currents to pass through circuits in a predetermined way. Six common logic gates are NOT, AND, OR, NAND, NOR, and XOR.

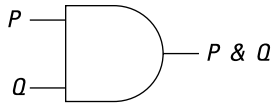
For example, here's a diagram for a NOT gate that shows the input (P) on the left and the output (~P) on the right:

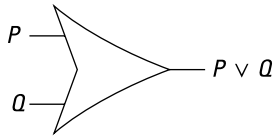


Here are some specifics about NOT gates:

- **In Boolean terms, a NOT gate changes an input of 0 to an output of 1, and an input of 1 to an output of 0.** In other words, when the current flowing into the gate is on, the current flowing out of it is off; when the current flowing in is off, the current flowing out is on.
- **The NOT gate is the only logic gate that has just one input.** This single input parallels the not-operator in that it's the only unary operator in SL. (By *unary*, I mean that the not-operator is placed in front of *one* constant rather than between *two* constants.)

The remaining logic gates all have two inputs. For example, take a look at these diagrams (with inputs on the left and outputs on the right) for an AND gate and an OR gate, respectively:





An AND gate turns its output current on only when both of its sources of input current are on; otherwise, its output current remains off. And, as you may have guessed, an OR gate outputs current when either of its input sources are on; otherwise, its output remains off.

REMEMBER

Even though the NOT, AND, and OR gates are sufficient for imitating any possible SL truth function, having a greater variety of building blocks is helpful for practical reasons. The following three gates are also commonly used:

- **NAND:** Short for *not and*, this gate behaves in a similar fashion to the nand–operator (see Chapter 13). Its output is 0 only when both of its inputs are 1; otherwise, its output is 1.
- **XOR:** Pronounced "eks-or," and short for *exclusive or*, this gate behaves like the exclusive *or* function I discuss in Chapter 13. That is, when exactly one of its two inputs is 1, its output is also 1; otherwise, its output is 0.
- **NOR:** Short for *not or*, this gate turns on its output current only when both of its inputs are off; otherwise, its output current remains off.

Computers and logic gates

Logic gates are the building blocks of a computer's *central processing unit* (*CPU*), which is the part of a computer where data is manipulated. Data is stored in memory devices and moved into and out of the CPU as needed, but the real "thinking" that goes on in a computer happens in the CPU.

As Alan Turing's work showed (see the earlier section "Turing and his universal Turing machine"), a short list of simple data manipulations is sufficient to calculate everything that can possibly be calculated. And logic gates provide more than enough of the basic functionality for reproducing a UTM.

Believe it or not, Charles Babbage's analytical engine (introduced in the section "Babbage designs the first computers" earlier in this chapter), which was designed to run programs on punch cards, would have no more and no less computing capability than today's state-of-the-art computers. Of course, today's computers have tons more memory and carry out all processes with lightning speed compared with the Babbage model. But, in principle, both machines can perform every possible mathematical calculation.

Comprehending software and computer languages

Computer hardware alone is far more complex than just about any machine you can name. Still, no matter how complex the basic machinery of a computer is, it's still limited to performing whatever functions it was designed to perform — just like any other machine.

For some applications, these limited capabilities are all that are needed. Computer circuits for cars, watches, and household appliances, for example, do a fine job of regulating the machines they're built into. But if you tried to use the circuit board from a BMW to run your dishwasher — or even a different type of car — you'd be disappointed with the results.

So why is your iPad or PC — or even your smartphone — able to perform a seemingly endless number of tasks? The answer, of course, is software.

REMEMBER

Software is any computer program that tells the hardware which tasks to carry out. All software is written in one of many *computer languages*, such as Java, JavaScript, Python, Prolog, and Ada (named in honor of Ada Lovelace, whose work I mention in the section "Babbage designs the first computers" earlier in the chapter). Even though all computer languages have differences in *syntax* (that is, the specific way in which commands are arranged in that language) and exhibit various strengths and weaknesses, they all have one thing in common: logic.

As with SL, computer languages allow you to declare variables and constants to which you can then assign values. For example, suppose you want to declare a variable called *month* and set its initial value to 10. Here are the lines of computer code you'd write in three languages:

- **JavaScript:** let month = 10;
- **Python:** month = 10
- **Prolog:** Month = 10.

As the lines of code show, the syntax is different in each language, but the main idea is the same. After you have variables to work with, you can build statements that test whether certain conditions are true or false — and then you can take action based on the results.

For example, suppose you want to send a message if and only if the date is October 9. Here's how you can write an *if*-statement in each of the three languages to test whether the month and day are correct, and, if so, how to perform the proper action:

» **JavaScript:**

```
if (month === 10 && day === 9) {
    message = "Happy Birthday!";
}
```

» **Python:**

```
if month == 10 and day == 9:
    message = "Happy Birthday!"
```

» **Prolog:**

```
birthday_message(10, 9, "Happy Birthday!").
birthday_message(_, _, "").
```

Again, the syntax differs from language to language, but the meaning is essentially the same.

Notice how the structure of a programming language enables you to use a computer to do, in a more streamlined fashion, what a UTM can do only in very tiny steps. Nevertheless, the basic idea behind both types of computers is the same, consisting of the following steps:

1. Set initial conditions.
2. Test the current conditions and make appropriate changes as needed.
3. Repeat Step 2 as often as necessary until the task is complete.

Ushering in Artificial Intelligence

Artificial intelligence (AI) represents a major leap in the evolution of computing.

For decades, computers were essentially rigid tools: They followed clear instructions and performed well-defined tasks, but lacked flexibility or independent

problem-solving ability. AI, by contrast, allows computers to behave more like collaborators — able to interpret natural language, solve complex problems, and even create new content.

In this section, I explore three ways AI is reshaping computing: the way humans provide input, what kinds of problems machines can solve, and the type of human-like outputs they can now produce.

Natural language: Speaking to machines on human terms

One of AI's most noticeable breakthroughs is its ability to work with natural language used by humans. In the past, interacting with a computer required learning *its* language — specific commands, codes, or menu structures. Now, with tools like ChatGPT and Google Gemini, users can simply ask questions or make requests in everyday language: "Write a summary of this article on the Human Genome Project," or "Explain who Julius Caesar was in words a fifth-grader can understand."

This shift — having a machine that understands human language — makes computing more accessible for everyone. You no longer need to be a programmer to get sophisticated help from a computer. AI systems can recognize context, adjust tone, and respond in ways that feel remarkably human. This conversational ability opens the door for broader use — from students seeking help with homework, to professionals writing reports or analyzing data.

Problem-solving: Handling greater breadth and depth

AI allows computers to tackle problems that go far beyond the scope of traditional software in both breadth and depth.

» **By *breadth*, I mean the capacity to synthesize diverse sources of information.** AI can integrate text, images, sound, and structured data to form insights that humans may miss. For example, AI can read medical records, interpret X-rays and CT scans, and suggest possible diagnoses. No single human, even a skilled professional, can do all three at the same scale or speed.

- **By *depth*, I'm referring to AI's ability to perform detailed analyses in specialized domains.** For example, when playing games such as chess or Go, AI systems have surpassed even the best human players, not by copying known strategies but by discovering new ones through self-training. AI is also used in science, finance, and engineering to model complex systems and make predictions far beyond what earlier software was able to achieve.

These advances in problem-solving ability have placed AI at the forefront of current computer technology.

Creative output: Mimicking human expression

Perhaps the most surprising capability of modern AI is its ability to generate original, humanlike content. Today's AI can write essays, craft poems, compose music, generate artwork, and even create videos, often with astonishing quality. Tools such as DALL·E (for images), MusicLM (for music), and Copilot (for computer code) demonstrate how AI can turn simple prompts from a human into polished results.

This creative potential is both exciting and controversial. That is,

- **It offers valuable assistance** by speeding up workflow, gathering input from vast resources, generating ideas, and helping people express themselves.
- **It raises ethical and philosophical questions,** such as
  - Who owns AI-generated content?
  - How do you prevent students from passing off AI-generated content as their own work?
  - What happens to artists and writers who should be credited and whose work is mimicked or replaced?

REMEMBER

Society will need to confront these issues as AI becomes increasingly integrated into creative industries.

IN THIS CHAPTER

- Introducing possibility with multi-valued logic and fuzzy logic
- Understanding modal, second-order, and paraconsistent logic
- Appreciating the mystery of quantum logic

Chapter 21

Sporting Propositions: Non-classical Logic

To most people, the question "What if 2 + 2 = 5?" is absurd. I mean, c'mon, asking yourself "What if . . . ?" in many cases is pointless because you know what you're pondering can never be true. For example, consider this wacky question: "What if little green Martians landed their flying saucer on my front lawn and drove off in my car?"

But, to logicians, these "What if?" questions are a *sporting proposition* — a challenge that's inviting just because it's so preposterous. By the early 20th century, logic had been reduced to a relatively short list of basic assumptions called *axioms*. (See Chapter 22 for more information about the axioms of logic.) These axioms were considered self-evident, and if you accepted them, everything else from logic followed.

But, what if you didn't accept them? What if, for the sake of argument, you changed an axiom the way a baker changes an ingredient in a cake recipe?

Of course, a baker should choose carefully when changing ingredients. For example, replacing baking powder with sugar would stop a cake from rising, and replacing chocolate with garlic would result in a cake that even your dog wouldn't eat.

Similarly, a logician needs to choose carefully when changing axioms. Even a small change can result in a logical system that's full of contradictions. And even if a system is consistent, it may prove to be trivial, uninteresting, or useless — the garlic cake of logic. But given these caveats, a few mavericks have still fashioned alternative systems of logic.

Welcome to the world of *non-classical logic*, the modern response to more than 2,000 years of *classical logic* (which includes just about everything I discuss in this book). In this chapter, I give you a taste of several non-classical systems of logic. These systems may not just challenge your notions of true and false but may also surprise you in how useful they turn out to be in the real world.

Opening Up to Possibility

In Chapter 1, I introduce you to the *law of the excluded middle* — which is the rule that a statement is either true or false, with no in-between allowed. This rule is an important assumption of logic, and for thousands of years it has served logic well.

When I introduce this law, I make it clear that not *everything* in the world fits together so neatly. Nor does it have to. The statement *Harry is tall* may be true when Harry is standing with a group of children, but it's likely false when he's standing next to LeBron James.

Typically, logicians acknowledge this black-and-whiteness to be a limitation of logic. Logic doesn't handle shades of gray, so if you want to work with this statement, you must agree on a definition for the word *tall.* For example, you may define a man who stands 6 feet or more in height as *tall,* and all others as *not tall.*

The law of the excluded middle was a cornerstone of logic from Aristotle well into the 20th century. It was one of those things you just accepted as a necessary assumption if you wanted to get anywhere with logic. But, in 1917, a man named Jan Lukasiewicz began wondering what would happen if he added a third value to logic. Check out the upcoming sections to see where Lukasiewicz's ponderings took him — and logic as a whole.

Three-valued logic

Lukasiewicz decided to see what would happen if a third value, one that was neither true nor false, was added to logic. He called this value *possible*, and he stated that people could assign it to statements whose truths are inconclusive, such as:

It will rain in Brooklyn tomorrow.

Doctors will someday find a cure for the common cold.

Life exists on other planets.

REMEMBER

Possible seems to exist somewhere between true and false. For this reason, Lukasiewicz started out with the Boolean representation of 1 for true and 0 for false. (See Chapter 14 for more about Boolean algebra.) He then added a third numerical value, ½, for possible.

The result was *three-valued logic.* Three-valued logic uses the same operators as classical logic, but with new definitions to cover the new value. For example, here's a truth table for $\sim x$:

x	$\sim x$
1	0
½	½
0	1

This value for possibility makes sense when you use it in an example, like this:

Let P = It will rain in Brooklyn tomorrow.

If the truth value for P is ½ — that is, if it's possible — it's also possible that it won't rain in Brooklyn tomorrow. So, the value of $\sim P$ is also ½.

By a similar line of reasoning, you can figure out the truth value of or-statements and and-statements that include one or more sub-statements whose values are possible. For example:

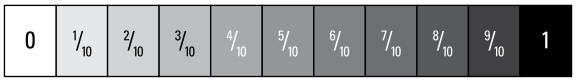
Either George Washington was the first president *or* it will rain in Brooklyn tomorrow.

This statement is true (the first part is true, so the rest doesn't matter), so its value is 1.

Multi-valued logic

REMEMBER

After the door swings open for an intermediate truth value, you can create a logical system with more than one intermediate value — in fact, any number you choose. For example, imagine a system with 11 values, from 0 to 1 with increments of $\frac{1}{10}$ in between. This system is an example of *multi-valued logic*.



How do you compute the truth value of statements with this system? In multi-valued logic, the rules are simple.

- **Negation-rule:** $\sim x$ means $1 - x$.
- **And-rule:** $x \,\&\, y$ means *choose the smaller value.*
- **Or-rule:** $x \vee y$ means *choose the larger value*.

These three rules may seem odd, but they work to create a new and consistent system of logic. And they include and extend three-valued logic (values of 0, 1, and ½), just as three-valued logic includes and extends Boolean algebra (values of 0 and 1).

For example:

Expression	Rule applied
$\sim 2/10 = 8/10$	**negation-rule**
$3/10 \,\&\, 8/10 = 3/10$	**and-rule**
$1/10 \vee 6/10 = 6/10$	**or-rule**
$7/10 \,\&\, 7/10 = 7/10$	**and-rule**
$9/10 \vee 9/10 = 9/10$	**or-rule**

TIP

You can even add in the other two sentential logic (SL) operators using the logical equivalence rules *implication* (**Impl**) and *equivalence* (**Equiv**). (See Chapter 10 for more information about equivalence rules.)

- **Impl:** $x \to y = \sim x \vee y$
- **Equiv:** $x \leftrightarrow y = (x \to y) \,\&\, (y \to x)$

With these multi-value logic rules, you can calculate the value of any expression step-by-step, much as you would write a proof. For example:

Expression	Rule applied
$4/10 \leftrightarrow 3/10$	
$= (4/10 \rightarrow 3/10) \,\&\, (3/10 \rightarrow 4/10)$	**Equiv**
$= (\sim 4/10 \vee 3/10) \,\&\, (\sim 3/10 \vee 4/10)$	**Impl**
$= (6/10 \vee 3/10) \,\&\, (7/10 \vee 4/10)$	**negation-rule**
$= 6/10 \,\&\, 7/10$	**or-rule**
$= 6/10$	**and-rule**

Lukasiewicz focused on the syntax of the system while leaving the semantics open to interpretation (see Chapter 14 for details on syntax and semantics). In other words, calculation in multi-valued logic is completely rule-based, but the meaning of your result is open to interpretation.

I describe several ways you can interpret results in the following section.

Fuzzy logic

REMEMBER

Some critics of multi-valued logic have questioned its usefulness to describe possible future events. They cite *probability theory* as the more appropriate tool for looking into the future. (Probability theory also calculates possibility, but it uses a different method of calculation.)

This criticism has some merit. For example, if the chance of rain in Brooklyn is $^{3}/_{10}$ and in Dallas is $^{7}/_{10}$, probability theory calculates the chance of rain in *both* places by multiplying these values.

$$3/10 \times 7/10 = 21/100$$

On the other hand, multi-valued logic would calculate this *and* value as $^{3}/_{10}$. Because probability theory is very well established, this discrepancy calls into question the usefulness of multi-valued logic.

In the 1960s, however, mathematician Lotfi Zadeh saw the potential for multi-valued logic to represent not what's possibly true but what's *partially true*.

Consider the following statement:

> I am hungry.

After you eat a big meal, this statement will most likely be false. If you don't eat for several hours, though, eventually it will be true again. Most people, however, don't experience this change as black-and-white, but rather as shades of gray. That is, they find that the statement becomes truer over time until they eat.

In fact, most so-called opposites — tall or short, hot or cold, happy or sad, naïve or worldly, and so forth — aren't separated by a sharp line. Instead, they're ends of a continuum that connects subtle shadings. In most cases, shadings of these types are somewhat subjective.

REMEMBER

Zadeh's answer to these issues is *fuzzy logic*, which is an extension of multi-valued logic. As with multi-valued logic, fuzzy logic permits a value of 0 for completely false, a value of 1 for completely true, or numerous values in between to describe shadings of truth. But, unlike multi-valued logic (which has a specified number of values between 0 and 1), fuzzy logic permits all values between 0 and 1 (an infinite number of values).

To understand fuzzy logic a bit better, consider a hypothetical couple, Donna and Jake, who have agreed to buy a brand-new television for their living room. Donna is thinking about a medium-sized model for around $500, but Jake has in mind a wall-covering giant that costs about $2,000. Of course, when they arrive at the store, the rude awakening occurs. If they look at their disagreement from the perspective of two-valued logic, they'll never make a decision.

But, after they start hashing things out, it's clear that some flexibility exists for both of them. Donna says that her absolute highest spending amount would be $1,500. Jake responds that he would want to go no lower than $800. You can represent these limits as follows, where $D(x)$ is Donna's truth value for x dollars, and $J(x)$ is Jake's:

Donna's limits	Jake's limits
$D(500)=1$	$J(800)=0$
$D(1500)=0$	$J(2000)=1$

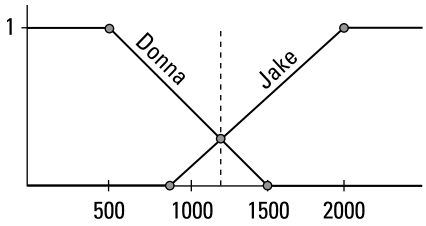


FIGURE 21-1: Compromising on the cost of a television.

With these four values, you can make a graph of the situation, as shown in Figure 21-1.

The point where the two lines cross, where x equals about $1,182, is the best compromise for the couple. So, to maximize their happiness with the outcome, Donna and Jake should try to find a television as close to this value as possible.

Getting into a New Modality

REMEMBER

Like multi-valued logic, *modal logic* attempts to handle not just the true and the false, but the possible. For this reason, modal logic introduces two new operators: the *possibility operator* and the *necessity operator*.

$\Diamond x$ = It is possible that x.

$\Box x$ = It is necessary that x.

For example:

Let C = Circles are round.

Then $\Diamond C$ means "It is possible that circles are round," and $\Box C$ means "It is necessary that circles are round."

The two modal operators are logically connected as follows:

$$\Box x = \sim \Diamond \sim x$$

For example, the statement "It is necessary that circles are round" is equivalent to the statement "It is not possible that circles are not round." Similarly, the following equation holds true:

$$\Diamond x = \sim \Box \sim x$$

For example, the statement "It is possible that circles are round" is equivalent to the statement "It is not necessary that circles are not round."

REMEMBER

One important aspect of modal logic is the distinction between *necessary truth* and *contingent truth*. To understand the distinction, consider the following example:

Let S = It snowed yesterday in Brooklyn.

Suppose that it really did snow yesterday in Brooklyn. In that case, both of these statements are true:

S	TRUE
$S \vee \sim S$	TRUE

Even though both statements are true, a difference between the types of truth exists.

- The first statement (S) is *contingently true* because its truth is contingent on what actually happened. That is, the situation might have been different. It's even possible that the weather report was inaccurate.
- The second statement ($S \vee \sim S$), however, is *necessarily true.* That is, regardless of the circumstances in Brooklyn yesterday, there's a logical necessity that the statement is true.

The shading of difference becomes clearer after I add the necessity operator.

$\Box S$	FALSE
$\Box(S \vee \sim S)$	TRUE

In this case,

- The first statement ($\Box S$) says: "It is necessary that it snowed yesterday in Brooklyn." In modal logic, this statement is false, because a scenario could exist in which it didn't snow there yesterday.
- The second statement ($\Box(S \vee \sim S)$) says: "It is necessary that it either snowed or didn't snow yesterday in Brooklyn." This statement is true, underscoring the stronger level of truth independent of real-world events.

HANDLING STATEMENTS WITH PROPOSITIONAL EMBEDDINGS

Modal logic is just one type of logic that attempts to include statements with shades of meaning that can't be translated into sentential logic and quantification logic. *Deontic logic* allows you to handle statements of obligation and permission. For example:

You *must* stop at a red light.

You *are allowed to* go through a yellow light.

Similarly, *epistemic logic* includes operators for handling statements of knowledge and belief. For example:

Arnie *knows that* Beth is waiting for him.

Arnie *believes that* Beth is waiting for him.

Modal, deontic, and epistemic logic are three examples of logic that includes *propositional embeddings* — that is, statements that are embedded within larger statement structures.

Taking Logic to a Higher Order

In Part 4 you discover that quantification logic (QL) includes individual constants, individual variables, and property constants, but not property variables. This setup allows you to quantify individuals, but not properties. For example, you can represent the sentence

All bankers are rich.

as

$$\forall x\,[Bx \rightarrow Rx]$$

To be specific, this statement tells you that anything that has the property of being a banker also has the property of being rich.

After you start focusing on properties themselves, however, you may want to discuss them in a logical fashion. For example, suppose you're hiring an assistant

and you're looking for someone who's friendly, intelligent, and upbeat. In QL, it's easy to declare constants for these properties as follows:

Let F = x is friendly.

Let I = x is intelligent.

Let U = x is upbeat.

Then you can represent the sentence

Nadia is intelligent and Jason is upbeat.

as

$In \,\&\, Uj$

However, with QL, you can't represent this statement:

Nadia has every property that Jason has.

REMEMBER

Second-order logic (also called *second-order predicate logic*) allows you to handle statements about properties that individuals possess by quantifying property variables rather than just individual variables. For example, suppose that X stands for some property. Having that variable set in place, you can now represent the previous statement as

$$\forall X\,[Xj \rightarrow Xn]$$

The literal translation of this statement is "For every property X, if Jason has the property X, then Nadia has the property X."

You can also quantify individual variables. For example, consider the following statement:

Someone has every property that Jason has.

You can represent this statement as

$$\forall X \exists y\,[Xj \rightarrow Xy]$$

The literal translation of this representation is "For every property X, there exists an individual y such that if Jason has the property X, then individual y has the property X."

Moving Beyond Consistency

REMEMBER

In a sense, *paraconsistent logic* is the flip side of multi-valued logic. In multi-valued logic, a statement may be *neither true nor false.* In paraconsistent logic, a statement may be *both true and false.* That is, every statement has at least one but possibly two truth values: **T** and/or **F**.

In other words, it's possible for a statement of the form x & ~x to be true. To put it another way, paraconsistent logic allows you to break the law of non-contradiction by permitting a statement to be both true and false. (See Chapter 1 for more on the law of non-contradiction.)

This essential difference between paraconsistent and classical logic creates further waves with one important rule of inference: *disjunctive syllogism* (**DS**). To refresh your memory about **DS**, check out Chapter 9 and consider these constants:

Let *A* = Ken lives in Albuquerque.

Let *S* = Ken lives in Santa Fe.

In classical logic, **DS** allows you to make the following inference:

$$A \vee S, \sim A : S$$

What this argument means is that if Ken lives in either Albuquerque or Santa Fe and he doesn't live in Albuquerque, then he lives in Santa Fe.

In paraconsistent logic, however, this inference doesn't hold true. Even though the failure of **DS** in paraconsistent logic may sound wacky at first, the reason makes sense when you think about it.

Suppose that both *A* and ~*A* are true, which is allowed in paraconsistent logic, no matter how odd it sounds. That is, Ken both *does* and *doesn't* live in Albuquerque. In that case, the statement $A \vee S$ is true even if *S* is false. And, as I just said, the statement ~*A* is also true. So, both of these statements are true, but the statement *S* is false. In other words, the two premises are true and the conclusion is false, so **DS** is an invalid argument in paraconsistent logic.

In this way, paraconsistent logic remains consistent even though, by definition, it's inconsistent. Or, putting it another way, paraconsistent logic is both consistent and inconsistent — which, when you think about it, is kind of the whole idea in a nutshell. Or not.

Making Logical Sense of Quantum Logic

Quantum computing is a revolutionary field of computer science that leverages the principles of quantum mechanics to perform computations in fundamentally new ways.

Quantum computers are not just faster versions of classical machines — they operate according to entirely different rules. This capability makes them especially suited for solving problems that are too large or complex to be solved with traditional computers.

At the heart of quantum computing is *quantum logic*, which is fundamentally different from classical logic and even the forms of non-classical logic I discuss earlier in this chapter. (See Chapter 20 for an explanation about the logic of classical computing.)

In this section, I introduce you to the genuine strangeness of quantum logic.

Breaking the law of the excluded middle

In Chapter 1, I discuss the *law of the excluded middle*, which states that every statement in classical logic is either true or false. So far in this chapter, you've seen a variety of ways that logicians have found to break this cornerstone law and yet still produce a functional system of non-classical logic.

REMEMBER

Quantum logic breaks the law of the excluded middle in a unique way that, while it may seem strange, is in accordance with the behavior of the physical world at the smallest known level — that is, the *quantum* level.

Unlike classical computers, which process information using bits that always represent one of the two Boolean values — either 0 or 1 (see Chapter 20) — quantum computers use quantum bits, or qubits, which can exist in a superposition of both 0 and 1 simultaneously. This key property makes quantum computers fundamentally different from classical computers and allows them to perform certain calculations much more efficiently than classical computers.

While classical logic is fundamentally binary, permitting two values (0 and 1), quantum logic is fundamentally nonbinary.

Looking at quantum logic differences

To help you understand how quantum logic differs from classical logic, here's a quick review of key facts about classical logic that hold true throughout most of this book:

- Classical logic is built on two truth values: true (1) and false (0).
- Every proposition is either true or false at all times — no third option exists.
- Logical operations are defined by using Boolean algebra, where values are manipulated using fixed truth tables (AND, OR, NOT, and so on).
- Classical computation mirrors this logic with bits that are always in one of two definite states.

This binary nature underpins everything from digital circuits to classical algorithms. (See Chapter 20 for more information about how classical computers rely on Boolean logic.)

By contrast, quantum logic breaks the binary mold in these key ways:

- Quantum logic permits a single *qubit* (the smallest indivisible atom of quantum computing) to be in a *superposition* of 0 and 1 — that is, neither 0 nor 1, but a combination of these two values based on probability — before a measurement takes place.
- A *measurement* is an operation that extracts a Boolean (binary) value from a qubit, collapsing its superposition into a value of either 0 or 1.
- Until a measurement is made, the qubit doesn't have a binary value.
- Measurement collapses the state to 0 or 1, but until then, the system exists in a continuum of states.

The presence of probability in a quantum system makes quantum logic essentially *non-deterministic*. That is, before a measurement is made, a qubit isn't simply in an *unknown* state that's equal to either 0 or 1; in fact, the qubit's state simply *cannot* be known until a measurement is made.

This non-determinism is the very feature of quantum mechanics that led Albert Einstein to reject it, saying skeptically, "Der Alte würfelt nicht." (This is often given the loose translation, "God does not play dice.")

Disturbing the distribution rule

In Chapter 10, you discover that in classical logic, the following two statements are semantically equivalent:

$$P \,\&\, (Q \vee R) \qquad\qquad (P \,\&\, Q) \vee (P \,\&\, R)$$

This equivalence is encapsulated as one of the two *distribution* (**Dist**) rules permitted in formal SL proofs.

In quantum logic, this rule no longer holds true. In this section, I help you understand **Dist**, so you can see why a world without it doesn't make sense at first. Then, I help you make sense of it.

Getting conned by the quantum shell game

Have you ever seen a con artist play the shell game? They show you a pea, place it on a table, and hide it under a walnut shell. Then they place two more empty shells on the table next to it and deftly move the three shells around on the table. If you can guess which shell the pea is hidden under, you win.

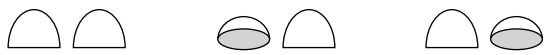
To get an idea of how quantum logic works, imagine a shell game with only two shells, where the peas are qubits. Now, begin with the understanding that the following statement is true:

> **Statement #1:** The pea is under either the left shell or the right shell.

REMEMBER

In quantum logic, many of the basic assumptions of classical logic are left intact. A statement may have one of only two possible values: **T** or **F**. Similarly, the basic operators from SL work just the same. So, in quantum logic, just as in SL, you can declare variables like these, as illustrated in Figure 21-2:

> Let *P* = The pea is under either the left shell or the right shell.
>
> Let *Q* = The pea is under the left shell.
>
> Let *R* = The pea is under the right shell.



The pea is under one of these two shells.

The pea isn't under the left shell.

The pea isn't under the right shell.

FIGURE 21-2: A mind-bending shell game.

With this structure in place, you can write a statement in English and translate it into an SL statement. For example, Statement #1 translated into quantum logic becomes this:

$$P \,\&\, (Q \vee R)$$

So far, so good: Everything up until now has been the same as in SL. Now, if you're using SL, you can use **Dist** (see Chapter 10) to write an equivalent statement. But, in quantum logic, **Dist** doesn't apply, so you can't rewrite the previous statement as this:

$$(P \,\&\, Q) \vee (P \,\&\, R) \qquad \text{WRONG!}$$

But, before you toss this statement aside, notice that it translates back into English in this way:

> **Statement #2:** *Either* the pea is under either the left shell or the right shell and it is under the left shell, *or* the pea is under either the left shell or the right shell and it is under the right shell.

In other words, Statement #1 and Statement #2 aren't necessarily equivalent, nor does one necessarily imply the other. So, it's possible for Statement #1 to be true and Statement #2 to be false. That is, it's true that the pea is under one of the two shells, but it isn't under the left shell and it isn't under the right shell.

Understanding how the quantum shell game really works

A key to getting a better grasp on how the quantum shell game works is to let go of the idea that the pea, when out of sight, is *really* under one shell or the other.

That concept (the pea must be under one of the shells) works well in classical logic, which underpins classical physics and classical computers. But in quantum logic, this assumption doesn't hold true.

A more accurate way to look at the quantum shell game is to understand that the pea isn't strictly under one shell to the exclusion of the other. Instead, it's under a combination of both shells at the same time in a way that can be understood with complete mathematical formality.

TIP

The math itself is a little hairy, and beyond the scope of this book. But if you're interested, I suggest checking out *Quantum Computing For Dummies*, by whurley and Floyd Earl Smith (Wiley).

IN THIS CHAPTER

- Grasping the basics of set theory and Russell's paradox
- Seeing how the SL axiomatic system measures up
- Appreciating consistency and completeness
- Limiting mathematics with Gödel's incompleteness theorem

Chapter 22
Paradox and Axiomatic Systems

Consider the following statement:

> This statement is false.

If the statement is true, then it must be false. However, if it's false, it must be true. This problem, called the *liar paradox,* dates back to the Greeks.

At first glance, the liar paradox seems no more than a curiosity. But paradoxes like this one have surfaced and resurfaced in various forms, causing trouble for logicians and challenging them to search for ways to resolve the trouble.

In this chapter, I explain Russell's paradox (a modification of the liar paradox), which forced logicians to undertake a radical restructuring of the foundations of set theory and logic. This paradox leads to a discussion of *Principia Mathematica*, which is an attempt to formulate set theory, logic, and ultimately all of math

based upon a set of assumptions called *axioms*. You also see how logic fares against the ultimate tests of mathematical certainty. And you catch a glimpse of the limits of what can be proven logically as I introduce Gödel's incompleteness theorem — arguably, the most important result in 20th-century mathematics.

Grounding Logic in Set Theory

Gottlob Frege's formulation of logic in the late 19th century depended on the relatively new work of Georg Cantor called *set theory*. Set theory is a remarkably simple way to organize objects in the real world. It also provides a unified framework to define mathematical objects, such as numbers, by their mathematical properties.

In this section, I show you how set theory provided a natural foundation for logic, how this foundation was threatened, and how it was made solid again.

Setting things up

Set theory deals with, not surprisingly, sets. A *set* is simply a collection of things. For example:

Let S = the set of all the shirts I own

Let H = the set of all the hats you own

REMEMBER

A set is properly defined only when you can clearly distinguish what's *in* the set and what's *not*.

The items in a particular set are called *elements* of that set. For example, the shirt I'm wearing right now is an element of set S, and your favorite hat (assuming you own it) is an element of set H.

Sets may contain other sets, called *subsets*. For example:

Let B = the set of all the blue shirts I own

Let L = the set of all the shirts I own that are in the laundry basket

Both B and L are subsets of S. That is, any element that's in either of these sets is also in set S.

REMEMBER

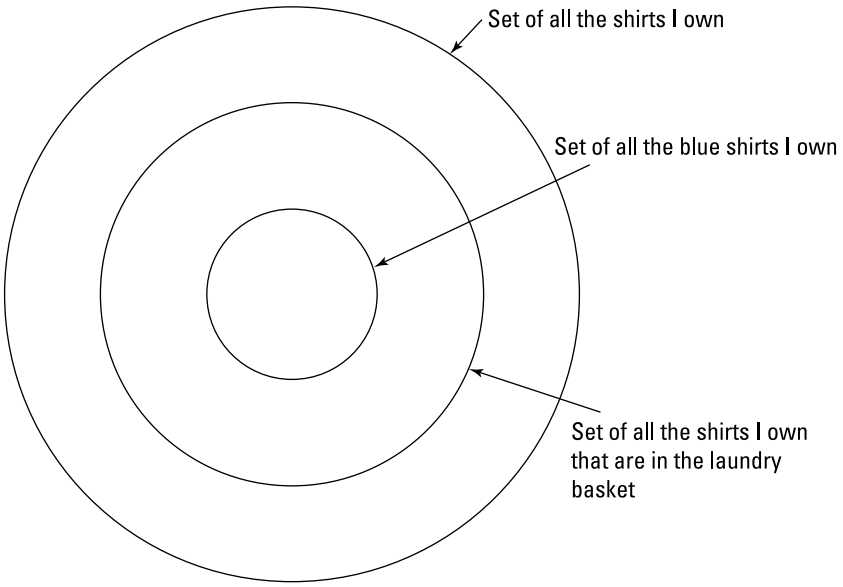
Even though set theory may seem rather simplistic, it's actually a powerful way to express logical ideas. For example, consider the following statement:

> All my blue shirts are in the laundry basket.

This statement is easily expressible in quantification logic (QL) as

$$\forall x[Bx \rightarrow Lx]$$

This statement is true if and only if set B is a subset of set L.



Although set theory and logic look different on the surface, they express similar ideas. That's why Frege formally grounded his logic in the apparently solid ground of set theory. As long as set theory was free from contradictions, logic was presumed consistent as well. And set theory was so simple that it seemed unassailable.

Trouble in paradox: Recognizing the problem with set theory

It took a genius, Bertrand Russell, to spot a problem with set theory. This flaw, called *Russell's paradox* in his honor, hinges on the notion of self-reference. For example, the liar paradox is a paradox of self-reference: The root of the problem is that the statement speaks about itself.

Set theory faced a similar problem because it was possible for a set to contain itself as an element. Most of the time, however, sets don't contain themselves as elements. For instance, looking back to the shirt and hat example from the previous section, you can see that set *S* contains only shirts and set *H* contains only hats. But consider this set:

> Let *X* = the set of all the sets mentioned in this chapter

Set *X* contains sets *S*, *H*, *B*, and *L* as elements, of course, but it also contains *itself* as an element, because set *X* is mentioned in this chapter.

This isn't a problem by itself, but it quickly leads to one. Consider what would happen if you defined this set:

> Let *Z* = the set of all sets that don't contain themselves as elements

In this case, sets *S*, *H*, *B*, and *L* are elements of set *Z*, but set *X* isn't. But here's a question that cannot be easily resolved:

> Is set *Z* an element of itself?

The problem here is similar to the liar paradox: If set *Z* is an element of itself, then by definition it *isn't* an element of itself. And if set *Z* isn't an element of itself, then by definition it *is*.

Developing a solution in Principia Mathematica

Russell's paradox (see the preceding section) was more than just troubling. The paradox forced logicians to recast set theory and logic in a different way. The trick became figuring out how to keep the paradox from creeping in while still retaining the bulk of what was useful and descriptive in the original system of set theory.

In an attempt to resolve the paradox — as well as to create a solid foundation for mathematics — in the first decade of the 20th century, Bertrand Russell and Alfred North Whitehead wrote *Principia Mathematica.* This ambitious work was the first full-scale attempt to describe all of mathematics as a formal *axiomatic system* — an organization of mathematical ideas based on a small number of statements assumed to be true.

REMEMBER

The core of an axiomatic system is a short list of simple statements called *axioms.* Axioms are combined in specifically defined ways to derive a much larger set of statements called *theorems.* Russell and Whitehead carefully chose their axioms with these purposes in mind:

- Creating a system powerful enough to derive sophisticated statements about mathematics as theorems
- Avoiding *all* inconsistencies, such as Russell's paradox
- Showing that *all* possible mathematical truths can be derived as theorems

Russell and Whitehead were certainly successful in achieving the first goal. And, their system eliminated paradoxes of self-reference, such as Russell's paradox. However, whether *Principia Mathematica* could avoid *all* inconsistencies and provide a method to derive *all* of mathematics remained to be seen.

TECHNICAL STUFF

Even though the axiom set of *Principia* did solve the problem of Russell's paradox, it was awkward in practice, so it didn't catch on with mathematicians. Instead, a different set of axioms, the *Zermelo-Fraenkel axioms* (*ZF axioms*), solved the problem by distinguishing sets from more loosely defined objects known as *classes.*

Today, the term *set theory* usually refers to one of several versions of set theory based in the ZF axioms. On the other hand, simpler versions of set theory that don't attempt to prevent such paradoxes do exist. These versions are collectively called *naïve set theory.*

REMEMBER

Explaining how axiomatic systems work is a major goal of this chapter. I discuss this concept in greater detail in the sections that follow.

Discovering the Axiomatic System for SL

To give you a sense of how theorems are formally derived in sentential logic (SL), this section outlines the basic structure of the axiomatic system Russell and Whitehead developed for *Principia Mathematica.*

REMEMBER

In a nutshell, a formal axiomatic system has four requirements. All these requirements are cast in terms of sets, because in logic and all other mathematics at the most formal level, *everything* is cast in terms of sets. That's why getting the bugs out of set theory, as I describe in the earlier section "Grounding Logic in Set Theory," was so critical.

The four requirements of a formal axiomatic system are

- **Requirement #1:** A set of symbols
- **Requirement #2:** A set of rules for deciding which strings of symbols are well-formed formulas (WFFs)
- **Requirement #3:** A set of axioms
- **Requirement #4:** A set of rules for combining axioms and/or theorems to create new theorems

So, with those requirements in mind, you can see how SL fits the definition of an axiomatic system. Requirement #1 is fulfilled because SL contains a set of symbols — operators, constants, and parentheses (see Chapter 4). Requirement #2 is fulfilled with the set of rules for WFFs (see Chapter 14).

To satisfy requirement #3, here are the four SL axioms from *Principia Mathematica:*

1. $(x \vee x) \rightarrow x$
2. $x \rightarrow (x \vee y)$
3. $(x \vee y) \rightarrow (y \vee x)$
4. $(x \vee y) \rightarrow ((z \vee x) \rightarrow (z \vee y))$

As far as requirement #4, take a look at the following two rules for making new SL theorems:

- **Rule of Substitution:** In each case, you can substitute a constant or an entire statement for a variable, as long as you do so uniformly throughout.

  For example, two possible substitutions for axiom #1 are

  - $(P \vee P) \rightarrow P$
  - $((P \,\&\, Q) \vee (P \,\&\, Q)) \rightarrow (P \,\&\, Q)$
- **Modus ponens (MP):** If you know that $x \rightarrow y$ and x are both theorems, you may add y to your list of theorems. See Chapter 9 for more on **MP**.

From this short list of rules, it's possible to derive all the rules of inference for SL, which I cover in Part 3. This shows you that, even though this list of axioms is short, the theorems it allows you to derive are numerous and powerful.

Proving Consistency and Completeness

In Chapter 6, you find out that a *tautology* is a statement that's always true. With the formalization of SL as an axiomatic system came two important proofs about SL: In SL, every theorem is a tautology, and every tautology is a theorem. That is, theorems and tautologies are equivalent in SL.

Throughout this book, you use both truth tables and proofs indiscriminately to draw conclusions about SL statements. After you know that every theorem is a tautology and vice versa, you have license to apply both *syntactic methods* (methods based on structure and arrangement, like proofs) and *semantic methods* (analyses focused on meaning and interpretation, like truth tables) to a specific problem. (You can test whether a particular theorem is a tautology by simply making a truth table and checking it with the method I discuss in Chapter 6.)

Still, when you step back a moment, you should realize that the equivalence of theorems and tautologies in SL isn't to be taken for granted.

- Why isn't it the case, for example, that a theorem can turn out not to be a tautology?
- Why isn't there a tautology that can't be produced as a theorem using the limited axioms and rules in the previous section?

The first question deals with the consistency of SL, and the second deals with the completeness of SL. I discuss both of these in the following sections.

Consistency and completeness of SL and QL

In 1921, mathematician Emil Post proved that SL is both consistent and complete. An axiomatic system is *consistent* if and only if every theorem generated in that system is a tautology. An axiomatic system is *complete* if and only if every tautology can be generated as a theorem in that system.

REMEMBER

In practice, consistency differs from completeness in an important way: Consistency is absolutely essential to an axiomatic system, while completeness is not.

To understand the importance of this point, imagine a mathematical system in which $2 + 2$ equals both 4 and 5. This single inconsistency would ripple throughout the entire system, making it unusable. For example, if 4 and 5 were equal to each other, then $4 + 1$ would equal $5 + 1$, so 5 and 6 would also be equal. In fact, *all* numbers would be equal to each other.

This example illustrates why Russell's paradox (which I explain in "Trouble in paradox: Recognizing the problem with set theory" earlier in this chapter) was so disastrous for logic: Until this inconsistency could be removed from the system, set theory was functionally unworkable, which threatened the functionality of logic.

In contrast to consistency, which is absolutely essential in a formal axiomatic system, mathematicians viewed completeness as more like the Holy Grail: desirable but difficult to attain.

In fact, completeness was an unspoken goal of mathematics dating back to the Greeks. Euclid, for example, is honored as the founder of geometry even though geometry had been studied for hundreds or even thousands of years before him. (See Chapter 2 for more information about Euclid.) Euclid's great insight was that geometry could be founded on five axioms, and that from these, all other true statements about geometry could be derived as theorems.

The decade following Post's proof was a fruitful time for logic. In 1928, David Hilbert and Wilhelm Ackermann proved that QL is consistent. Then, in 1931, Kurt Gödel proved that QL is complete. This important result was his doctoral dissertation, the first work of the man who would become one of the greatest mathematicians of the 20th century.

Formalizing logic and mathematics with Hilbert's Program

By the 1920s, logic and mathematics had developed sufficiently for a precise examination of whether every mathematical truth could be shown as a theorem within a formal axiomatic system. Mathematician David Hilbert was instrumental in advocating the complete formalization of logic and mathematics. This formalization became known as *Hilbert's Program.*

Hilbert realized that what philosophers and mathematicians had been intuitively reaching for since the days of Aristotle and Euclid was now potentially in reach: a single axiomatic system, purged of inconsistencies, for expressing and calculating all logical and mathematical truths.

Hilbert's Program stressed the need for expressing all of mathematics in strict axiomatic terms. All logical assumptions must be stated explicitly in formal language lacking all ambiguity. (*Principia Mathematica* was an example of an attempt at such formalization, and Hilbert studied it in depth.) From this foundation, the intuitive notion of mathematical proof was formalized, resulting in *proof theory.*

PLAYING WITH PEANO'S AXIOMS

One important difference exists between the axioms of *Principia Mathematica* and the axioms of logic: The *Principia* axioms were strong enough to derive mathematically sophisticated statements. (See the earlier section "Developing a solution in *Principia Mathematica*" for more information about *Principia*.)

The *Principia* axioms made possible the derivation of five foundational axioms of number theory which is the foundation of all higher mathematics. The five axioms developed by mathematician Giuseppe Peano are

1. Zero is a number.
2. If *n* is a number, then the successor of *n* is a number.
3. Zero is not the successor of any number.
4. Two numbers whose successors are equal are themselves equal.
5. If a set *S* contains zero and the successor of every number, then every number is in *S*. (This is called the *induction axiom*.)

In 1931, Gödel showed that *Principia Mathematica*, an axiomatic system powerful enough to derive Peano's axioms (and thus model mathematics), was doomed to be either inconsistent or incomplete. More generally, he showed that *any* axiomatic system powerful enough to model mathematics was similarly doomed.

Hilbert championed the ongoing search for consistency and completeness proofs for axiomatic systems. Comparing an axiomatic system to a ship at sea, you can say that consistency means the ship won't sink, and completeness means the ship will take you everywhere you want to go.

Examining Gödel's Incompleteness Theorem

The proofs showing that QL was both consistent and complete made mathematicians optimistic that Hilbert's Program would succeed. Ironically, the man who proved the completeness of QL would soon after demonstrate that Hilbert's Program would never succeed.

In 1931, Kurt Gödel published his *incompleteness theorem*, which states that no axiomatic system can have all three of the following properties:

- **Consistency:** Every theorem of the system is a tautology in an area that the system intends to model.
- **Completeness:** Every tautology in an area that the system intends to model is a theorem of the system.
- **Sufficient power to model mathematics:** The system can be applied as a model for mathematics.

The importance of Gödel's theorem

Gödel's incompleteness theorem is generally considered to be the most important mathematical result of the 20th century. It's astounding because it places limits on what mathematics can do — or, more precisely, it limits the degree to which axiomatic systems can describe mathematics. And, in one coup de grâce, Gödel's incompleteness theorem demonstrated that the goal of Hilbert's Program (see the preceding section) was unattainable.

Interestingly, Gödel proved his conjecture by adopting a strategy along the same lines as the liar paradox and Russell's paradox: namely, self-reference. But, instead of being confounded by paradox, Gödel made paradox work in his favor. His strategy hinged on the fact that any system expressive enough to model complex mathematics would also be expressive enough to model itself — an extremely tricky task.

How he did it

As a first step, Gödel showed how to assign a unique number, called the *Gödel number*, to every string of symbols in a system. This numbering allowed him to uniquely number not only random strings but also statements, theorems, and even entire arguments, valid or otherwise. For example, all these strings can be represented by a Gödel number:

$$4 + 9 = 13$$

$$4 + 9 = 1962$$

$$\forall x \exists y [x + y = 3]$$

$$= 48 < +33 - = 7 =$$

Because Gödel numbering works at the level of the string, literally nothing that can be expressed in the system escapes his numbering, including that last string shown above, which is meaningless (or, in syntactic terms, not a well-formed statement).

Next, he showed how to use the system to build special statements called *meta-statements*, which refer to other statements. For example:

> The statement "$4+9=13$" is a theorem.
>
> The statement "$4+9=13$" is not a theorem.
>
> The statement "$4+9=13$" includes the symbol "3."
>
> There exists a proof that the statement "$4+9=13$" is a theorem.

These meta-statements themselves are just statements, each with its own Gödel number. In this way, one meta-statement can refer to another meta-statement. A meta-statement can even refer to itself. For example:

> There does not exist a proof that this statement is a theorem.

REMEMBER

The paradox in the previous statement is even more subtle than it appears because Gödel built the statement to guarantee that at the level of semantics, the statement is a tautology. (The math behind this guarantee is completely over the top, so just take my word for it.) But then consider the following:

- If the statement *can* be derived as a theorem, then it isn't a theorem, so the contradiction of the statement can also be derived as a theorem, making the system by definition *inconsistent.*
- If the statement *can't* be derived as a theorem, then the system is by definition *incomplete.*

If this line of thinking sounds complicated, that's because it is. And I've only scratched the surface. Understanding *what* Gödel accomplished is fascinating, but understanding *how* he accomplished it is something that many talented mathematicians have only attained in a more or less sketchy fashion.

Pondering the Meaning of It All

Since Gödel published his proof undermining the usefulness of axiomatic systems to fully express mathematical truth, opinions about its meaning on a philosophical level have been divided.

In a sense, Gödel's incompleteness theorem was a response to Gottfried Leibniz's 250-year-old dream of finding a system of logic powerful enough to calculate questions of law, politics, and ethics. (See Chapter 2 for more about Leibniz.) And Gödel's response was a definitive "No! You just can't do that!" Given that logic is insufficient for framing a complete model of mathematics, it certainly seems unlikely that it will ever adequately provide the tools to resolve ethical questions by mere calculation.

You may think Gödel's proof implies that the rational mind is limited in its ability to understand the universe. But, although the mind may have its limitations, Gödel's result doesn't prove that these limitations exist. The proof just discusses how axiomatic systems are limited in how well they can be used to model other types of phenomena. The mind, however, may possess far greater capacities than an axiomatic system or a *universal Turing machine* (see Chapter 20 for more on this important concept).

Another common, and probably hasty, reaction to Gödel's work is to assume that his proof implies a limit on artificial intelligence. After all, human intelligence has developed rather well in this universe. Why can't other forms of intelligence, even artificial ones, develop along similar lines?

REMEMBER

Like other uncanny scientific results of the 20th century — such as relativity theory and quantum mechanics — Gödel's proof answers one set of questions only to open up a new set of questions that are far more compelling.

6 The Part of Tens

IN THIS PART . . .

Ten quotes about logic from thinkers and cynics throughout the ages.

Ten nominees for the Logicians Hall of Fame.

Ten helpful tips for passing a logic exam.

IN THIS CHAPTER

- Introducing logic quotes for $2,000, please
- Getting some logical perspective

Chapter 23
Ten (or So) Quotes about Logic

Okay, so I actually include 11 quotes here — but who's counting? They're all interesting and offer a variety of perspectives on logic.

AMBROSE BIERCE — AMERICAN WRITER/SATIRIST

Logic: The art of thinking and reasoning in strict accordance with the limitations and incapacities of the human misunderstanding.

SAMUEL BUTLER — ENGLISH NOVELIST/ESSAYIST

Logic is like the sword — those who appeal to it shall perish by it.

G. K. CHESTERTON — ENGLISH WRITER

You can only find truth with logic if you have already found truth without it.

ALBERT EINSTEIN — GERMAN PHYSICIST

Logic will get you from A to B. Imagination will take you everywhere.

JOSEPH WOOD KRUTCH — AMERICAN NATURALIST/WRITER

Logic is the art of going wrong with confidence.

JOHN LOCKE — 17TH CENTURY ENGLISH PHILOSOPHER

Logic is the anatomy of thought.

KATE MULGREW — AMERICAN ACTOR (SPEAKING AS CAPTAIN KATHRYN JANEWAY ON *STAR TREK: VOYAGER*)

You can use logic to justify almost anything. That's its power — and its flaw.

LEONARD NIMOY — AMERICAN ACTOR (SPEAKING AS SPOCK ON *STAR TREK*)

Logic is the beginning of wisdom, not the end.

ANTOINE DE SAINT-EXUPÉRY — FRENCH WRITER

Pure logic is the ruin of the spirit.

GLORIA STEINEM — AMERICAN ACTIVIST/WRITER

Logic is in the eye of the logician.

LUDWIG WITTGENSTEIN — AUSTRIAN PHILOSOPHER

Logic takes care of itself; all we have to do is to look and see how it does it.

IN THIS CHAPTER

- Seeing how Aristotle changed it all
- Discovering the people who transformed logic
- Understanding the evolution of logic

Chapter 24
Ten Big Names in Logic

This chapter contains my nominees for the Logic Hall of Fame. I had to pass over lots of great minds so this list wouldn't be miles long, but these logicians are my top ten.

Aristotle (384–322 BCE)

Aristotle was the originator of logic. Before him, philosophers (such as Socrates and Plato) and mathematicians (such as Pythagoras and Thales) presented arguments on a wide variety of topics. But Aristotle was the first to examine the structure of argument itself.

REMEMBER

In a series of six philosophical writings on logic, later collected as a single work titled *Organon*, Aristotle identified the foundational concepts in logic. He defined a *statement* as a sentence that possesses either truth or falsehood (for more information about statements, flip to Chapters 1 and 3). He also studied valid argument structures called *syllogisms* (see Chapter 2), which contain premises that lead inevitably to a conclusion. For centuries after his death, Aristotle's writings on logic were often studied and commented upon, but rarely surpassed. (See Chapter 2 for more fun facts about Aristotle.)

Peter Abelard (1079–1142)

Peter Abelard stands out as the preeminent figure of medieval logic for his groundbreaking contributions that reshaped the field. Unlike his predecessors — who largely preserved Aristotelian syllogistics — Abelard pioneered a shift toward propositional logic by examining how whole statements, not just terms, convey truth.

He was the first to explore aspects of modal logic, analyzing necessity and possibility long before the field matured. His work on semantics — how words and sentences generate meaning — laid the foundation for later theories of reference and signification. In his book *Sic et Non*, Abelard introduced a dialectical method that encouraged critical thinking and rational analysis. This method transformed how logic was taught and applied. More than a transmitter of logic, Abelard was an original thinker who opened new paths that others would follow for centuries. (Flip to Chapter 2 for more on the work of Abelard and other Medieval logicians.)

Gottfried Leibniz (1646–1716)

A bona fide Renaissance man, Gottfried Leibniz was the first philosopher in the Age of Reason to see the potential for logic to be used as a tool for calculation. He hoped that logical calculation would someday be on par with mathematics. He even worked out the beginnings of a symbolic representation of logic, anticipating formal logic by 200 years. (Check out Chapter 2 to read more about Leibniz.)

George Boole (1815–1864)

George Boole invented Boolean algebra, which was the prototype for formal logic. Boolean algebra was the first system of logic that relied on pure calculation to determine the truth value of statements. Boole used 1 to represent true and 0 to represent false. Computer scientists still use Boolean variables as objects that can take only these two values and no others. (See Chapter 2 for more information about Boole and Chapter 14 for a look at Boolean algebra.)

Georg Cantor (1845–1918)

Georg Cantor was the inventor of set theory, which was the foundation of mathematical logic and, arguably, for all other mathematics. He was also the first to incorporate into math an understanding of infinity as a calculable entity instead of just a mysterious quality of *never-endingness*. For all these achievements and more, Cantor is on everybody's short list for the greatest mathematician of the 19th century. (Flip to Chapter 2 for more information about Cantor and Chapter 22 for more about set theory.)

Gottlob Frege (1848–1925)

Gottlob Frege, inventor of formal logic, built upon the work of Boole, Cantor, and others to develop the first logical systems recognizable as what were to become sentential logic (see Part 2) and quantification logic (see Part 4). His logic included the five logical operators *not*, *and*, *or*, *if*, and *if and only if*. It also included symbols for *all* and *there exists*. (Check out Chapters 2 and 22 to read more about Frege and his contribution to logic.)

David Hilbert (1862–1943)

David Hilbert was tremendously influential in the fields of logic and mathematics. He advocated for the rigorous reduction of all mathematics to *axioms* (self-evident truths) and *theorems* (statements that can be logically proved from axioms). This trend in mathematics became known as *Hilbert's Program*, whose goal became the creation of an axiom system for mathematics that was both consistent and complete — that is, it produced all possible true theorems and no false ones.

Although Kurt Gödel proved that the goal of Hilbert's Program wasn't attainable, Hilbert's contribution to the development of mathematical logic is undeniable. (See Chapter 22 for more information about Hilbert.)

Bertrand Russell (1872–1970)

Bertrand Russell's nearly 100-year lifespan included such notable achievements as discovering Russell's paradox and coauthoring *Principia Mathematica* with Alfred North Whitehead.

REMEMBER

Russell's paradox resulted in a reformulation of Frege's logic and Cantor's set theory — both foundational systems that had previously appeared unassailable. *Principia Mathematica* was Russell's attempt to formulate mathematics with perfect logical consistency and completeness. (See Chapters 2 and 22 for further discussion about Russell and his place in the history of logic and mathematics.)

Kurt Gödel (1906–1978)

Kurt Gödel proved that logic, in its most powerful form (quantification logic), is mathematically both consistent and complete. *Consistency* means that logic is free from contradictions. *Completeness* means that all true logical statements can be proved by syntactic methods. (See Chapter 7 for a discussion of syntactic proof.)

Gödel is most famous, however, for his proof that mathematics as a whole does *not* possess consistency and completeness. He demonstrated that any mathematical system that's free from contradictions must contain statements that are true but can't be proved as true using the axioms of that system. This discovery signaled the end of Hilbert's Program and is generally considered the most important mathematical result of the 20th century. (See Chapters 2 and 22 for more details about Gödel's important work.)

Alan Turing (1912–1954)

Alan Turing proved that all calculations that humans perform can be equally accomplished by a computer having a specified set of simple functions. These functions include the ability to check whether certain types of conditions are true or false and to take action based on the result.

REMEMBER

Turing called any computer of this kind a *universal Turing machine (UTM)*. Because every modern computer is a UTM, and logic is at the heart of how any UTM processes data, logic is a cornerstone of computer science. (Flip to Chapter 20 for more on how logic and computer science work together.)

IN THIS CHAPTER

- Discovering techniques to help you effectively manage a logic exam
- Figuring the quickest way out of a logic jam
- Understanding the importance of checking your work and admitting your mistakes

Chapter 25
Ten Tips for Passing a Logic Exam

So, let me guess — you're frantically reading this book eight hours before your big logic final? Not to worry. This short chapter offers ten tips to help you excel at exam time. Read on — the grade you save may be your own!

Breathe

As you sit down and wait for your prof to hand out the exam, or before the timer on your digital exam platform starts, take a deep breath in to a slow count of five and then breathe out in the same way. Repeat. That's all. Do this for a minute or so (but no longer than that — you don't want to hyperventilate!). You'll find that you're now much calmer and ready to tackle the exam.

Start by Glancing over the Whole Exam

It takes only a minute, but looking through the entire test (or at least through the questions in a single module of a digital exam) gets your brain working subconsciously on the problems as early as possible. As a result, a few of the problems may work themselves out more easily as you go along.

This technique also allows you to find helpful clues. For example, Question 3 may ask you to define a term that appears in Questions 5, 6, and 7.

Warm Up with an Easy Problem First

Why shouldn't you warm up with an easy problem? Athletes know the importance of a warm-up — for example, by performing easy stretches — to get everything moving. And I guarantee that just getting at least one question behind you will make you feel better instantly.

Fill In Truth Tables Column by Column

When filling in a truth table, you can do it the easy way or the hard way. The easy way is to fill in the **T** or **F** values column by column, as I show in Chapter 6.

If you try to go row by row instead, you'll have to keep switching gears (that is, you must switch between determining values for constants and finding them for operators), and completing the truth table will take longer.

If You Get Stuck on a Proof, Jot Down Everything

Because proofs look so neat in the textbook, some students think they have to figure them out in their heads before they begin delivering their proof path.

But writing proofs is a messy process, so feel free to make a mess. Use scratch paper to write down every possible path you can think of. After the big "Aha!" happens and the path becomes clear, start writing the proof neatly on the page (or platform) you intend to turn in.

If You REALLY Get Stuck on a Proof, Move On

It happens: The answer is staring you square in the face and you can't see it. The minutes are whizzing by, your heart is pounding, and your palms are so sweaty that your pencil (or mouse) is dripping.

If you're the praying kind, this would be a good time to start asking for divine guidance. But remember, the Lord helps those who help themselves, and so I say unto thee: Moveth on!

Better to miss this one question than the five you never answered because you got stuck. If you have time, you can always go back to the troublesome question. And when you do, the answer may just jump out at you.

If Time Is Short, Finish the Tedious Stuff

Generally speaking, truth tables and truth trees tend to be plug-and-chug methods: Slow and steady work will always get you to the finish line. Quick tables and proofs, on the other hand, tend to be creative methods: You can solve them quickly if you have the right insight, but you may get bogged down if your brain just isn't clicking.

So, when the professor calls out, "Ten minutes left!" (and they really mean it), or your exam platform's digital timer shows ten minutes remaining, put aside the ornery proof that just isn't speaking to you at the moment and pick up the tedious truth table you've been avoiding for half the exam. As the exam grinds to its close, you're more likely to make progress if you work on the straightforward table instead of just staring at the impenetrable proof.

Check Your Work

I'm aware that in certain circles, checking your work on an exam is considered an outdated practice — like carrying a monogrammed hanky or removing your hat upon entering a building.

But think about the last time you *didn't* check over your exam before handing it in. Did you really do something more important with those seven or eight minutes you saved? No, you probably spent those precious minutes texting with your friends from class to express your collective relief that the exam was over.

And don't you just hate getting your exam back *after* it's been graded and catching those silly errors only because they're circled in red or otherwise highlighted?

All right, so I'm sneezing into the wind here. But, even if you catch only one little 3-point mistake, you may bump yourself up a partial letter grade. And when you take the time to look over your answers (if you have that extra time), catching 5-, 10-, or even 20-point mistakes isn't that uncommon.

Admit Your Mistakes

I know that this piece of advice flies in the face of all your survival instincts, but that's why I'm passing it on.

Consider this tragic scenario: You've sweated for half an hour over a hairy question trying to prove $P \& Q$. As your time runs out, you get to the last line and find you've proved $\sim(P \& Q)$. Ouch! If you had time, you could go through your proof line by line and find the error, but time is exactly what you don't have.

Your first instinct — maybe the prof won't notice, and they'll give you full credit — is dead wrong. (Your second instinct, which may be to quit school and move to a Tibetan monastery, also probably isn't a good idea.) If you leave the mistake as is, your professor will think you don't know the difference between a statement and its negation.

Instead, you should somehow mark the error and write a note: "I know this is wrong, but no time to fix it." Now when reading through the proof, the professor has some context. If you made a minor mistake, such as dropping a not-operator (~), they'll probably just ding you for a point or two. Even if you really messed up the proof, at least the prof knows you're aware of it. So, now they know you're not completely clueless and you're rather honest as well. Either way, you win!

Stay Focused Until the Bitter End

Keep answering questions and reviewing your answers until the professor pries the pencil from your blistered fingers or your digital exam platform shuts down your efforts. You're bound to pick up a few points in those last few minutes or catch a mistake (if you're just reviewing), and the prof will notice your extra effort.

Index

SYMBOLS

– (minus sign), 76
∀ quantifier. *See* universal quantifier
:: (double-colon symbol), 179
| (nand-operator), 234–235, 334
= (equals sign), 243
× (multiplication symbol), 243
∨ operator. *See* disjunction operator
→ operator. *See* if-operator
↔ operator. *See* if-and-only-if-operator
∃ quantifier. *See* existential quantifier
& operator. *See* conjunction operator
~ operator. *See* negation operator

A

Abelard, Peter, 24–25, 374
accent, fallacy of, 49–50
Ackermann, Wilhelm, 364
ad hominem argument, 42
addition (Add) implication rule, 170–171, 215, 278
Age of Faith, 26
Age of Reason, 26
AI (artificial intelligence), 33, 337–339
algebra, Boolean, 27–28, 65, 242–246
algorithms, 331–332
"all" statements, 264–267, 269, 272, 283
all-or-nothing argument, 45
ambiguity, fallacies of, 48–50
amphiboly, fallacies of, 49
analytical engine, 330, 336
AND logic gate, 334–335
and operator. *See* conjunction operator
and-rules, 163, 344–345
and-statement (conjunction), 97
antecedent, if-statements, 9
AP. *See* assumed premise
appeal to authority, 44
appeal to belief, 43
appeal to emotion, 43
appeal to ignorance, 44
appeal to purity, 49
Aquinas, Thomas, 23, 25
arguments. *See also* logical fallacies
  bridging premises-conclusion divide, 162–163
  cogent, evaluating, 56–57
  conclusion in, 13
  enthymemes in, 13–14
  examples of, 39–41
  intermediate steps in, 13
  logic in relation to, 36
  overview, 12, 35
  premises in, 12
  reasonable doubt, 56
  rhetoric, 55–57
  sound, 53
  structure of, 36–38
  valid, 13, 36, 38–39, 113–116, 118–119
Aristotelian logic, 18–21, 23–24, 373
Aristotle, 1, 39–40, 59
Art of Discovery, The (Leibniz), 26
artificial intelligence (AI), 33, 337–339
association (Assoc) equivalence rule, 185–186, 215, 221–222, 226
assumed premise (AP)
  combining conditional and indirect proofs, 205–206
  conditional proofs, 197–198, 213, 215
  discharging, 198, 200–202, 206, 215
  indirect proofs, 203–205
  stacking assumptions, 201–202
  undischarged, 299
  working backward from conclusion, 219

assumptions. *See also* conditional proofs; indirect proofs
  fallacies of unwarranted, 47–48
  hidden, in arguments, 13–14
  shrewd, when handling difficult proof, 226–227
  strategic, and quick tables, 124, 126–129
authority, appeal to, 44
Avicenna, 23–24
axiomatic systems
  consistency and completeness, proving, 363–365
  Gödel's incompleteness theorem, 365–368
  Hilbert's Program, 364–365
  overview, 357–358
  in Principia Mathematica, 360–361
  for sentential logic, 361–362
axioms, 16, 22, 161, 365, 375

B

Babbage, Charles, 329–330, 336
bandwagon fallacy, 48
begging the question, 46
belief, appeal to, 43
biconditional operator. *See* if-and-only-if-operator
Bierce, Ambrose, 371
binary operators, 76
black-and-white thinking, 16
Boethius, 23–24
Book of Healing, The (Avicenna), 24
Boole, George, 26, 334, 374
Boolean algebra, 27–28, 65, 242–246
bound variables, in QL, 260–261
brackets
  with existential quantifier, 256
  instantiation rules and, 283
  role in translating categorical statements, 264, 267
  scope of quantifiers and, 260
  with universal quantifier, 255
  variables and, 261–262
branches, truth trees. *See* truth trees
brute force approach to truth tables, 102
Buridan, Jean, 25–26
Butler, Samuel, 371

C

Cantor, Georg, 28–29, 358, 375
categorical statements, 20, 263–273
Categories (Aristotle), 19
cause and effect, 9
CD (constructive dilemma) implication rule, 174–175
central processing unit (CPU), 335
ChatGPT, 33, 338
Cheat Sheet for book, 4
cherry-picking, 46
Chesterton, G. K., 371
Chrysippus, 22–23
circular reasoning, 47
classical logic
  Aristotle, 18–21
  Chrysippus and Stoics, 22–23
  Euclid, 22
  Jean Buridan, 25–26
  medieval logic, 23–24
  overview, 18
  Peter Abelard, 24–26
  quantum logic differences from, 352–353
  Thomas Aquinas, 25
  William of Ockham, 25
cogent arguments, evaluating, 56–57
commutation (Comm) equivalence rule, 184–185, 215, 221–222
commutative property, 184
completeness
  defined, 376
  of quantification logic, 363–364
complex statements in SL, translating, 81
composition, fallacy of, 50
computer logic
  algorithms, 331
  artificial intelligence, 337–339
  early versions of computers, 330–333

evolution of modern computer, 333
hardware and logic gates, 333–336
overview, 32–33, 58, 329
programs, 332
software and computer languages, 336–337
truth functions, 333–334
computing truth value. *See* evaluation
conclusion
bridging premises-conclusion divide, 162–163
comparing to premises, 208
in logical arguments, 13
noticing length of, 208
tweaking with equivalence rules, 199–201
when choosing between direct and indirect proof, 217
working backward from in difficult proofs, 218–220
conditional operator. *See* if-operator
conditional proofs (CPs)
combining indirect proof and, 205–206
moderate proofs, handling with, 211–216
overview, 195–197
stacking assumptions, 201–202
steps in, 197–199
tweaking conclusion for, 199–201
conditional statements, 22
conjunction (Conj) implication rule, 167–169, 278–279
conjunction operator
defined, 65
implication rules for and-statements, 167–170
incorporating into quantification logic, 253
replacing with other operators, 233–234
in sentential logic, 68–69
translating statements with, 79, 82
conjunction statement, 97
conjunctions, 81
consequent, if-statements, 9
consistency
consistent statements, 101
defined, 376
evaluating with truth tables, 111–113
of sentential logic, 363–364
showing with truth tree, 141–144
testing for with quick table, 128–129
Consolation of Philosophy, The (Boethius), 24
constants
in Boolean algebra and SL, 243
existential instantiation with, 291
property, 251–253, 309
in quantification logic, 250–253
replacing with truth values, 90
statement, 64–65
truth tables, 104
universal generalization with, 296
constructive dilemma (CD) implication rule, 174–175
context, creating with domain of discourse in QL, 257–259
contingent statements, 109, 128, 153, 324
contingent truth, 348
contradictions
building, 119
building new, 116–119
contradictory pairs, 21
defined, 324
disproving assumption, 126
dividing statements into, 109
indirect proof, 203
separating with truth trees, 151–153
testing for with quick table, 127–128
contraposition (Contra) equivalence rule, 180–181, 199–200, 210
contrapositive of statement, 73
converse of statement, 72–73
Copilot, 33, 339
correlation-causation fallacy, 43
CPs. *See* conditional proofs
CPU (central processing unit), 335
creative output by AI, 339

D

DALL·E, 339
De Morgan's theorem (DeM) equivalence rule
  breaking down long premises, 225
  conditional proofs, 213, 215
  converting all negative statements with, 210–211
  distribution and, 224
  downsizing operators, 233–234
  overview, 188–190
  working backward from conclusion, 219
decomposition of statements, 138–140, 144–145, 147–152, 316–317
deduction, 54–55
deontic logic, 349
Descartes, René, 26
difference engine, 330
difficult proofs, handling
  breaking down long premises, 224–226
  choosing between direct and indirect proof, 216–218
  digging deeper into SL statements, 220–224
  making shrewd assumption, 226–227
  overview, 216
  working backward from conclusion, 218–220
dilemma, false, 45
direct proof, 216–218. *See also* equivalence rules; implication rules
discharging the AP, 198, 200–202, 206, 215
disjunction operator, 253
  defined, 65, 69–71
  implication rules for, 170–172
  replacing with other operators, 233–234
  in sentential logic, 69–71
  translating statements with, 79, 83
disjunctive syllogism (DS) implication rule, 171–172, 188, 209, 226, 277
distribution (Dist) equivalence rule, 186–188, 220, 222–224, 353–355
division, fallacy of, 50
DN (double negation) equivalence rule, 179–180, 281
domain of discourse, 257–259, 264–267
double implication, 97, 191–193
double negative statements in SL, 67
double-branching statements, 138–139, 145, 151–152, 319
double-colon symbol (**::**), 179
DS implication rule. *See* disjunctive syllogism

E

easy proofs, handling, 208–211
effect, cause and, 9
EG. *See* existential generalization
EI. *See* existential instantiation
Einstein, Albert, 353, 371
Electronic Numerical Integrator and Computer (ENIAC), 333
emotion, appeal to, 43
enthymemes, 13–14
equals sign (=), 243
equivalence (Equiv) rule
  breaking down complex-looking statements, 210
  conditional proofs, 214
  downsizing operators, 232
  multi-valued logic, 344–345
  overview, 191–193
  working backward from conclusion, 218
equivalence rules
  applying in QL, 279–280
  association, 185–186, 221–222
  commutation, 184–185, 221–222
  contraposition, 180–181, 210
  De Morgan's theorem, 188–190, 210–211
  distribution, 186–188, 222–224
  double negation, 179–180
  equivalence, 191–193
  exportation, 183–184, 221
  implication, 181–183, 210
  implication rules versus, 178
  overview, 177, 179
  tautology, 190–191
  tweaking conclusion with, 199–201
equivocation, fallacies of, 49

Euclid, 22, 364
evaluation, 89–99. *See also* truth tables
exams, tips for passing, 378–379
excluded middle argument, 45
excluded middle, law of, 15, 352
exclusive *or*, 71
existence, explaining, 11
existence statements, 11
existential generalization (EG)
  defined, 275, 284
  with multiple quantifiers, 310
  overview, 283–284, 287–290
  universal generalization similarity to, 295
existential instantiation (EI)
  defined, 275, 284
  example using, 291–292
  limits on using, 292–293
  with multiple quantifiers, 309
  overview, 283–284, 290
  similarity to universal instantiation, 290–291
  in truth trees, 317–319
  valid and invalid uses of, 294–295
  with variables and constants, 291
existential quantifier (∃)
  alternative translations of basic forms, 268–271
  alternative translations of basic forms of categorical statements, 268–269
  applying rules with multiple quantifiers, 308–310
  contradictory statements, 267–268
  existential generalization, 283–284, 287–290
  existential instantiation, 283–284, 290–295
  identifying categorical statements, 271–273
  "no" statements, 273
  overview, 256
  proving QL statements with, 278–279
  quantifier negation, 280–283
  quantifier rules overview, 283–284
  with relations, 304–305
  restricted domain, 265
  "some" statements, 272
  translating basic categorical statements, 264–268
  translating from English into QL, 267
  unrestricted domain, 265–267
  working with multiple quantifiers, 305–306
exportation (Exp) equivalence rule, 183–184, 221, 224

F

fallacies, logical. *See* logical fallacies
fallacies of presumption, 47
fallacy of accent, 49–50
fallacy of composition, 50
fallacy of division, 50
false dilemma, 45
false equivalence, 47
first-order logic. *See* quantification logic
first-order predicate logic. *See* quantification logic
formal logic, 2, 27–30, 63, 242. *See also* sentential logic
free variables, in QL, 260–261
Frege, Gottlob, 28–31, 358, 375
friendly forms, conditional proofs, 212
fuzzy logic, 345–347

G

gambler's fallacy, 45
generalization, 10–11, 44
generalization rules, 283–284, 287–290, 295–300
geometry, Aristotle's use of, 18
Gödel, Kurt, 32, 364–368, 375–376
Google Gemini, 338
grammar, 238. *See also* syntax
gut strategy for easy proofs, 208–211

H

hardware, computer, 32, 58, 333–336
hasty generalization, 44
hidden assumptions, 13–14
Hilbert, David, 364, 375

Hilbert's Program, 364–365, 375
human need to make sense of world, 8–9
hypothetical syllogism (HS) implication rule, 173–174, 296

I

ID (rule of identity), 312–313
identities, in QL, 301–302, 310–314
identity, law of, 14–15
identity reflexivity (IR), 313–314
if-and-only-if-operator
  defined, 65, 74–75
  quantification logic, 253
  replacing with other operators, 232
  in sentential logic, 74–75
  translating statements with, 80
if-and-only-if–rules, 173–175, 296
if-and-only-if–statements, 97, 191–193
if-operator
  defined, 65
  quantification logic, 253
  replacing with other operators, 233
  in sentential logic, 71–73
  translating statements with, 80, 83
if-rules
  modus ponens, 164–166, 210, 277, 362
  modus tollens, 166–167, 180, 210, 224, 289–290, 309
if-statements, 97
  antecedent, 9
  consequent, 9
  defined, 9
  double if-rules, 173–175
  implication and, 181
  implication rules for, 164–167
  in practical logic, 9–10
  sub-statements, 9
  turning or-statements into, 200–201
ignorance, appeal to, 44
implication (Impl) equivalence rule
  breaking down long premises, 224–225
  conditional proofs, 214
  distribution and, 223
  downsizing operators, 233
  easy proofs, handling, 210
  multi-valued logic, 344–345
  overview, 181–183
  tweaking conclusion with, 200–201
  working backward from conclusion, 219
implication rules
  addition, 170–171
  conjunction, 167–169
  constructive dilemma, 174–175
  disjunctive syllogism, 171–172, 209
  equivalence rules versus, 178
  hypothetical syllogism, 173–174
  modus ponens, 164–166, 210
  modus tollens, 166–167, 210
  overview, 161–164
  simplification, 169–170, 209
  transferring into QL, 277–279
inclusive *or*, 71
incompleteness theorem, 32, 365–368
inconsistency
  converting into contradiction, 118
  inconsistent statements, 111
  showing with truth tree, 141–144
  testing for with quick table, 128–129
indirect discourse, 311, 349
indirect proof
  choosing between direct proof and, 216–218
  combining conditional proof and, 205–206
  defined, 22
  overview, 195, 202–204
  proving short conclusions, 204–205
individual constants, in QL, 251–253
individual variables, in QL, 254–255
induction, 54–55

inference, rules of. *See* rules of inference
infinite trees in QL, 323–325
infinity, 375
input values, 76–77
instantiation rules. *See* existential instantiation; universal instantiation
intermediate steps, in arguments, 13, 36
interpretation of statement, 89, 101
invalid arguments, 113, 129, 144–146
inverse of statement, 73
IR (identity reflexivity), 313–314

J

JavaScript, 336–337
justification, proofs, 165

K

kitchen-sinking strategy, 182, 209
Krutch, Joseph Wood, 372

L

languages, computer, 336–337
law
  of excluded middle, 15, 342, 352
  of identity, 14–15
  of non-contradiction, 15
  use of logic in, 58–59
Leibniz, Gottfried, 26–27, 374
liar paradox, 357
loaded questions, 46
Locke, John, 372
logic. *See also* history of logic
  big names in, 373–376
  combining with math, 16
  common use of, 57–59
  deduction and induction in, 54–55
  formal logic, 2
  innate understanding of, 8–12
  misconceptions about, 50–51
  and nature of reality, 52
  overview, 1–4, 7, 35
  philosophical origin, 1–2
  quotes about, 371–372
  in relation to arguments, 36
  thinking versus, 51–52
  tips for passing exam on, 377–381
  transition to computation, 2
  types of, 17
  words used when making logical connections, 11–12
logic gates, 32–33, 333–336
logical arguments. *See* arguments
logical fallacies
  of ambiguity, 48–50
  overview, 35, 41–42
  of presumption, 45–47
  of relevance, 42–43
  of unwarranted assumptions, 47–48
  of weak induction, 43–45
logical operators
  comparing math operators and, 76–77
  main, finding, 94–96
  naming by input values, 76
  scope of, 93–94
  in sentential logic, 65–75, 229–236, 253–254
logically equivalent statements, 75
Lovelace, Ada, 330, 336
Lukasiewicz, Jan, 31, 342–343, 345

M

main operators, 94–96, 99
Martin, Steve, 237
math
  combining logic and, 16
  formalizing with Hilbert's program, 364–365
  operators in, 76–77
  similarity of evaluation to, 88–89
  similarity of sentential logic to, 75–78
  use of logic in, 57

medieval logic, 23–24
middle-ground fallacy, 48
minus sign (–), 76
modal logic, 24–25, 347–349, 374
moderate proofs, using conditional proof for, 211–216
modern logic, 26–30
modus ponens (MP) implication rule, 164–166, 210, 277, 362
modus tollens (MT) implication rule, 166–167, 180, 210, 224, 289–290, 309
monadic expressions, 302–304
Mulgrew, Kate, 372
multiplication symbol (×), 243
multi-valued logic, 31, 344–345
MusicLM, 339

N

naïve set theory, 361
NAND logic gate, 335
nand-operator (|), 234–235, 334
natural language, AI ability to work with, 338
necessary truth, modal logic, 348
necessity operator, modal logic, 347
need to make sense of world, 8–9
negated conclusion, 217
negation operator
  defined, 65–67
  identifying scope of, 93–94
  quantification logic, 253
  in sentential logic, 66–67
  translating statements with, 79, 82–83
negation rule, multi-valued logic, 344–345
negative forms, SL statements, 96
neither . . . nor structure, translating into SL, 82–83
Newton, Isaac, 26
Nimoy, Leonard, 372
"no" statements, 267–268, 271, 273, 283
no true Scotsman fallacy, 49
nominalism, 25
non-classical logic
  fuzzy logic, 345–347
  modal logic, 347–349
  multi-valued logic, 344–345
  overview, 31–32, 341–342
  paraconsistent logic, 350–351
  quantum logic, 351–355
  second-order logic, 349–350
  three-valued logic, 342–343
non-contradiction, law of, 15
non-determinism, 349, 353
non-statements, distinguishing statements from, 37–38
non-terminating trees in QL, 323–325
NOR logic gate, 335
not . . . both structure, translating into SL, 83
"not all" statements, 267–268, 270–271, 273, 283
NOT logic gate, 334
not operator. *See* negation operator

O

Ockham, William of, 25
Ockham's (Occam's) razor, 25
On Interpretation (Aristotle), 19
. . . only if . . . structure, translating into SL, 83–84
operators. *See* logical operators
oppositions, square of, 20–21
. . . or . . . but not both structure, translating into SL, 84–85
. . . or . . . or both structure, translating into SL, 84
. . . or . . . structure, translating into SL, 84
OR logic gate, 334–335
Organon (Aristotle), 18–19, 41, 373
or-operator. *See* disjunction operator
or-rules, 163, 344–345
or-statements, 97, 181, 200–201
output values, 76–77

P

paraconsistent logic, 350–351
paradoxes, 357–361. *See also* axiomatic systems

parallel statements, 187, 189
parentheses, 77–78, 90, 95–96
particular statements, 20
Peano, Giuseppe, 365
philosophy, use of logic in, 59
Plato, 18
Popper, Karl, 55
positive forms, SL statements, 96
possibility operator, modal logic, 347
Post, Emil, 363
post hoc, ergo propter hoc, 47–48
Posterior Analytics (Aristotle), 19
postulates. *See* axioms
predicate logic. *See* quantification logic
premises, 36. *See also* assumed premise; proofs
  in arguments, 12
  bridging premises-conclusion divide, 162–163
  comparing to conclusion, 208
  long, breaking down in difficult proofs, 224–226
  noticing length of, 208
presumption, fallacies of, 45–47
Principia Mathematica (Whitehead and Russell), 31–32, 357–358, 360–361, 376
Prior Analytics (Aristotle), 19
probability theory, 345
problem-solving in AI, 338–339
programs, computer, 332
Prolog, 336–337
proof by contradiction. *See* indirect proof
proof theory, 364
proofs. *See also* conditional proofs; equivalence rules; implication rules; quantification logic
  breaking down long premises, 224–226
  bridging premises-conclusion divide, 162–163
  choosing between direct and indirect proof, 216–218
  digging deeper into SL statements, 220–224
  easy, handling, 208–211
  friendly forms, 212–214
  indirect proofs, 195, 202–206
  making shrewd assumption, 226–227
  overview, 161–162, 207, 216
  slightly-less-friendly forms, 214–216
  tips for handling on exams, 378–379
  unfriendly forms, 216
  working backward from conclusion, 218–220
property constants, 251–253, 309
property variables, 255
propositional logic, 374. *See also* sentential logic
purity, appeal to, 49
Pythagoras, 18
Python, 336–337

Q

QL. *See* quantification logic
QN. *See* quantifier negation
qualitative sciences, 58
quantifiable sciences, 58
quantification logic (QL)
  applying SL rules, 276–280
  completeness of, 363–364
  creating context with domain of discourse in, 257–259
  defined, 30
  existential generalization, 287–290
  existential instantiation, 290–295
  existential quantifier in, 256
  identities in, 301–314
  individual variables in, 254–255
  overview, 249–250, 275
  quantifier negation, 280–283
  quantifier rules overview, 283–284
  relations in, 301–310
  SL operators in, 253–254
  statements and statement forms in, 259–262
  translation, 263–273
  truth trees in, 315–325
  types of constants in, 251–253
  universal generalization, 295–300
  universal instantiation, 284–287
  universal quantifier in, 255–256
  writing with identities, 312–314
  writing with relations, 307–310

quantifier negation (QN)
defined, 275
overview, 280–283
in truth trees, 317–319
using with multiple quantifiers, 307–308
quantifiers, QL. *See also* existential quantifier; universal quantifier
order of, 305–306
translating categorical statements, 264–268
quantum logic, 351–355
quick tables
basic steps to using, 123–126
overview, 121–123
planning strategy for, 127–129
reading, 125
truth tables versus, 122
truth trees versus, 138
working smarter with, 130–136

R

range of influence, operator's scope, 93
reality, 52–53
reasonable doubt, 56
reasoning, circular, 47
red herring, 42–43
reductio ad absurdum. *See* indirect proof
relations, in QL, 301–310
relaxation rule, WFF, 240–241
relevance, fallacies of, 42–43
Renaissance, 26–27
restricted domain of discourse, when translating into QL, 265
rhetoric, 14, 55–57
rows, truth tables, 104
rule of identity (ID), 312–313. *See also* identities, in QL
rules of inference. *See* equivalence rules; implication rules
Russell, Bertrand, 14, 31, 359–361, 376
Russell's paradox, 31, 359–360

S

Saint-Exupéry, Antoine de, 372
science, use of logic in, 57–58
scope
of logical operators, 93–94
of quantifier, determining, 260
second-order logic, 349–350
self-referential statements, 309
semantic equivalence
changing into tautology, 117
checking for with truth trees, 153–157
comparing SL to Boolean algebra, 242–246
judging with truth tables, 109–111
semantically equivalent statements, 101
testing for with quick table, 128
semantic inequivalence, 128, 153–157
semantics, 24, 237–242, 362, 374
comparing SL to Boolean algebra, 242–246
semi-rings, 245
sentential logic (SL). *See also* evaluation; proofs; quick tables; truth tables; truth trees
applying rules from to QL proofs, 276–280
axiomatic system for, 361–362
comparing to Boolean algebra, 242–246
consistency of, 363–364
defined, 29
logical operators in, 65–75
overview, 64
quantification logic versus, 249–250
similarity to math, 75–78
statement constants in, 64–65
statement forms, 96–98
translation, 78–85
truth value in, 65
set theory, 28–29, 358–361
Sheffer, Henry, 234
Sheffer stroke, 234–235, 334
Sic et Non (Abelard), 25, 374
"sidestepping the issue," 46
simplification (Simp) implication rule, 169–170, 209, 223, 278–279

single-branching statements, 138–139, 145, 151, 318
SL. *See* sentential logic
slightly-less-friendly forms, conditional proofs, 212
slippery slope argument, 46
slothful induction fallacy, 44–45
social sciences, 58
Socrates, 18
software, computer, 33, 58, 336–337
"some" statements, 264–267, 270, 272, 283
Sophistical Refutations (Aristotle), 41–42
sorites, 10
sound arguments, 53
sporting propositions, 341. *See also* non-classical logic
square of oppositions, 20–21
statement constants, 64–65
statement forms, 259–262, 279, 289–290
statements. *See also* evaluation; proofs; translation; truth tables
  in arguments, 37–38
  categorical, 20, 264–273
  comparing to QL statements, 276–277
  decomposing, 138–140, 144–145, 147–152
  in difficult proofs, 220–224
  forms of SL statements, 96–98
  identifying SL sub-statements, 92
  main operators in SL statements, 94–96
  in quantification logic, 259–262, 276–277
  repeated chunks of, 209
  scoping out SL statements, 93–94
  SL statements in quick tables, 130–136
  well-formed formulas, 238–242
Steinem, Gloria, 372
Stoics, 22–23
strategic assumptions
  argument validity and invalidity, 129
  consistent statements, 128–129
  contradictions, 127
  disproving, 126
  planning quick table strategy, 127–129
  quick tables as beginning with, 124
  semantic equivalence and inequivalence, 128
  tautology, 127
straw man fallacy, 42
strings, syntax of, 238–242
sub-statements, 9, 92, 138–140
substitution, 77, 362
Summa Logicae (William of Ockham), 25
Summa Theologica (Aquinas), 25
Summulae de Dialectica (Buridan), 25
suppressed evidence, 46
syllogisms, 39–40, 173–174, 296, 373
symbolic logic, 26–27. *See also* formal logic
syntactic methods, 362
syntax
  comparing SL to Boolean algebra, 242–246
  defined, 237
  overview, 237–239
  well-formed formulas, 238–242

T

tables. *See* quick tables; truth tables
tautologies, 52
  building, 116–119
  dividing statements into, 109
  separating with truth trees, 147–150
  testing for with quick table, 127
tautology (Taut) equivalence rule, 190–191
Thales, 18
theorems, 16, 22, 361, 375
thinking, versus logic, 51–52
thought, laws of, 14–15
three-valued logic, 342–343
translation
  quantification logic, 263–273
  sentential logic, 78–85
trunk, truth trees, 140
truth functions, 333–334

truth tables
brute force approach, 102
building new tautologies and contradictions, 116–119
consistency across statements, 111–113
dividing statements into categories with, 109
evaluating validity with, 113–116
filling in, 105–108
judging semantic equivalence with, 109–111
overview, 67, 101
quick tables versus, 122
reading, 108
setting up, 103–105
testing for logical conditions with, 116
tips for handling on exams, 378
truth trees versus, 138
using instead of quick tables, 135
truth trees
and consistency or inconsistency, 141–144
contingent statements, separating, 153
contradictions, separating, 151–153
decomposing SL statements, 138–140, 144–145, 147–152
overview, 137–138
in quantification logic, 315–325
and semantic equivalence or inequivalence, 153–157
solving problems with, 140–141
tautologies, separating with, 147–150
using instead of quick tables, 136
validity or invalidity, testing for with, 144–146
truth value, 38, 65, 99. *See also* evaluation
tu quoque (whataboutism), 48
Turing, Alan, 331–333, 335, 376
20th century, logic in, 31–33

U

UG. *See* universal generalization
UI. *See* universal instantiation
unary operator, 76, 334
undecidable statements, 32
unfriendly forms, conditional proofs, 212
universal generalization (UG)
defined, 275, 284
example using, 296–297
limits on using, 297–299
with multiple quantifiers, 310
overview, 283–284, 295
similarity to EG, 295
valid and invalid uses of, 299–300
with variables and constants, 296
universal instantiation (UI)
applying more than once, 319–323
defined, 275, 284
overview, 283–287
similarity to existential instantiation, 290–291
in truth trees, 317–319
universal quantifier (∀)
"all" statements, 272
alternative translations of basic forms, 268–271
alternative translations of basic forms of categorical statements, 268–269
applying rules with multiple quantifiers, 308–310
contradictory statements, 267–268
identifying categorical statements in disguise, 271–273
"not all" statements, 273
overview, 255–256
proving QL statements with, 278–279
quantifier negation, 280–283
quantifier rules overview, 283–284
with relations, 304–305
restricted domain, 265
translating basic categorical statements, 264–268
translating from English into QL, 267
universal generalization, 283–284, 295–300
universal instantiation, 283–287
unrestricted domain, 265–267
working with multiple quantifiers, 305–306
universal statements, 20

universal Turing machine (UTM), 331–333, 376
unpacking statements, 169, 189, 191–193
unrestricted domain of discourse, 265–267
unwarranted assumptions, fallacies of, 47–48

V

validity
  of arguments, 13, 36, 38–39, 113–116, 118–119
  evaluating with truth tables, 113–116
  testing for with quick table, 129
  testing for with truth tree, 144–146
variables
  Boolean, 334
  domain of discourse, 257
  existential instantiation with, 291
  existential quantifier and, 256
  quantification logic, 250
  in quantification logic, 254–255, 260–261
  universal generalization with, 296
  universal quantifier and, 255

W

weak induction fallacies, 43–45
well-formed formulas (WFFs), 238–242
whataboutism (*tu quoque*), 48
Whitehead, Alfred North, 31, 360–361, 376
William, of Ockham, 23
Wittgenstein, Ludwig, 372

X

XOR logic gate, 335

Z

Zadeh, Lotfi, 346
Zermelo-Fraenkel axioms (ZF axioms), 361

About the Author

Mark Zegarelli is the author of *Basic Math & Pre-Algebra For Dummies* (Wiley), *SAT Math For Dummies* (Wiley), *Fractions For Dummies* (Wiley), and eight other books in the *For Dummies* series. He holds degrees in both English and math from Rutgers University. Most recently, he provides online lessons for kids as young as four years old, showing them easy ways to understand — and enjoy! — square numbers and square roots, factors, prime numbers, fractions, and even basic algebra concepts.

Dedication

李光耀，我最喜欢的朋友

Author's Acknowledgments

This is my 11th *For Dummies* book, and again I enjoy the privilege of working with an editorial team that continues to inspire and call me to my best. Thank you to my Wiley editors: executive editor Lindsay Berg, associate editor Alicia Sparrow, project manager and development editor Leah Michael, copy editor Kelly Brillhart, managing editor Kristie Pyles, technical editor Amy Nicklin, and production editor, Tamilmani Varadharaj.

Publisher's Acknowledgments

Associate Acquisitions Editor: Alicia Sparrow
Project Manager: Leah Michael
Copy Editor: Kelly Brillhart
Technical Editor: Amy Nicklin
Production Editor: Tamilmani Varadharaj
Cover Images: © earthphotostock/Shutterstock, © Urupong/stock.adobe.com

WILEY END USER LICENSE AGREEMENT

Go to www.wiley.com/go/eula to access Wiley’s ebook EULA.